Wing IDE Reference Manual
Wing IDE Professional

Wingware
www.wingware.com

Version 2.1.4
February 9, 2007

Contents

Introduction

1.1. Product Levels
1.2. Licenses
1.3. Supported Platforms
1.4. Supported Python versions
1.5. Technical Support
1.6. Prerequisites for Installation
1.7. Installing
1.8. Running the IDE
1.9. Installing your License
1.10. User Settings Directory
1.11. Upgrading
1.11.1. Fixing a Failed Upgrade
1.12. Advanced Installation
1.12.1. Installing Extra Documentation
1.12.2. Linux Installation Notes
1.12.3. Source Code Installation
1.13. Removing Wing IDE
1.14. Command Line Usage
1.15. Trouble-shooting
1.15.1. Trouble-shooting Failure to Start
1.15.2. Issues on Microsoft Windows
1.15.3. Trouble-shooting Failure to Debug
1.15.3.1. Failure to Start Debug
1.15.3.2. Extra Debugger Exceptions
1.15.3.3. Failure to Stop on Exceptions
1.15.3.4. Failure to Stop on Breakpoints or Show Source Code
1.15.4. Obtaining Diagnostic Output
1.15.5. Speeding up Wing
1.15.6. Trouble-shooting Failure to Open Filenames Containing Spaces
1.15.7. Trouble-shooting Failure to Print

Customization

2.1. User Interface Options

2.1.1. Windowing Policies
2.1.2. User Interface Layout
2.1.3. Altering Text Display
2.1.4. Setting Overall Display Theme
2.2. Preferences
2.2.1. Preferences File Layers
2.2.2. Preferences File Format
2.3. Editor Personalities
2.4. Key Equivalents
2.4.1. Key Names
2.5. File Sets
Project Manager
3.1. Creating a Project
3.2. Removing Files and Packages
3.3. Saving the Project
3.4. Sorting the View
3.5. Keyboard Navigation
3.6. Sharing Projects
3.7. Project-wide Properties
Environment
Debug
Options
Extensions
3.8. Per-file Properties
File Attributes
Editor
Environment
Debug
3.9. Navigating to Files
Source Code Editor
4.1. Syntax Colorization
4.2. Right-click Editor Menu
4.3. Navigating Source

4.4. File status and read-only files
4.5. Transient vs. non-Transient Editors
4.6. Structural Folding
4.7. Brace Matching
4.8. Indentation
4.8.1. Auto-Indent
4.8.2. The Tab Key
4.8.3. Checking Indentation
4.8.4. Changing Block Indentation
4.8.5. Indentation Manager
4.9. Auto-completion
4.10. Source Assistant
4.11. Auto-save
4.12. Notes on Copy/Paste
4.13. Auto-reloading Changed Files
4.14. Search/Replace
4.14.1. Toolbar Quick Search
4.14.2. Keyboard-driven Mini-Search/Replace
4.14.3. Search Tool
Wildcard Search Syntax
4.14.4. Search in Files Tool
4.14.4.1. Replace in Multiple Files
4.15. User-defined Bookmarks
4.16. Templating (Code Snippets)
Overview
Syntax
Indentation and Line Endings
Cursor Placement
Reloading
Commands
User Interface
4.17. Using Revision Control with Wing
Installing CVS
Installing Subversion
Installing Perforce

Using SSH Repositories
Subversion with SSH
Subversion with http/https or file URLs
Subversion without Authentication Cache
Using CVS with SSH
Using CVS with pserver
Notes on the Implementation
4.18. Keyboard Macros
4.19. Source Code Analysis
4.19.1. Analysis Disk Cache
Source Code Browser
5.1. Display Choices
5.1.1. Viewing by Module
5.1.2. Viewing Class Hierarchy
5.1.3. Viewing Current Module
5.2. Display Filters
5.2.1. Filtering Scope and Source
5.2.2. Filtering Construct Type
5.3. Sorting the Browser Display
5.4. Navigating the Views
5.5. Browser Keyboard Navigation
Debugger
6.1. Quick Start
6.2. Specifying Main Entry Point
6.3. Debug Properties
6.4. Setting Breakpoints
Breakpoint Types
Breakpoint Attributes
Breakpoints Tool
Keyboard Modifiers for Breakpoint Margin
6.5. Starting Debug
6.6. Debugger Status
6.7. Flow Control

6.8. Viewing the Stack
6.9. Viewing Debug Data
6.9.1. Stack Data View
6.9.1.1. Popup Menu Options
6.9.1.2. Filtering Value Display
6.9.2. Tracking Values
6.9.3. Evaluating Expressions
6.9.4. Problems Handling Values
Stored Value Errors
6.10. Interactive Debug Probe
6.11. Interactive Python Shell
6.12. Managing Exceptions
6.13. Debug Process I/O
6.13.1. External I/O Consoles
6.13.2. Disabling Debug Process I/O Multiplexing
6.14. Attaching and Detaching
6.14.1. Access Control
6.14.2. Detaching
6.14.3. Attaching
6.14.4. Identifying Foreign Processes
6.14.5. Constraints
6.15. Debugging Externally Launched Code
6.15.1. Importing the Debugger
6.15.2. Debug Server Configuration
6.15.3. Remote Debugging
6.15.4. File Location Maps
6.15.4.1. File Location Map Examples
6.15.5. Remote Debugging Example
6.15.6. Debugger API
6.16. Running Without Debug
6.17. Debugger Limitations

Scripting and Extending Wing IDE

7.1. Scripting Example
7.2. Getting Started
Naming Scripts

Reloading Scripts
Overriding Internal Commands
7.3. Script Syntax
Script Attributes
ArgInfo
Commonly Used Types
Commonly Used Formlets
Magic Default Argument Values
GUI Contexts
Top-level Attributes
Importing Other Modules
Internationalization and Localization
7.4. Scripting API
7.5. Advanced Scripting
Example
How Script Reloading Works
Preferences Reference
User Interface
Files
Editor
Debugger
Source Analysis
IDE Extension Scripting
Internal Preferences
Core Preferences
User Interface Preferences
Editor Preferences
Project Manager Preferences
Debugger Preferences
Source Analysis Preferences
Source Browser Preferences
Command Reference
Top Level Commands

Dock Window Commands
Document Viewer Commands
Editor Browse Mode Commands
Editor Insert Mode Commands
Editor Non Modal Commands
Editor Panel Commands
Editor Replace Mode Commands
Editor Split Commands
Editor Visual Mode Commands
Global Documentation Commands
Toolbar Search Commands
Window Commands
Wing Tips Commands
Active Editor Commands
General Editor Commands
Project Manager Commands
Project View Commands
Debugger Commands
Debugger Watch Commands
Search Manager Commands
Search Manager Instance Commands

License Information

10.1. Wing IDE Software License
10.2. Open Source License Information

Wingware, the dancing bird logo, Wing IDE, Wing IDE Personal, Wing IDE Professional, Wing IDE Enterprise, and "Take Flight!" are trademarks or registered trademarks of Wingware in the United States and other countries.

Disclaimers: The information contained in this document is subject to change without notice. Wingware shall not be liable for technical or editorial errors or omissions contained in this document; nor for incidental or consequential damages resulting from furnishing, performance, or use of this material.

Hardware and software products mentioned herein are used for identification purposes only and may be trademarks of their respective owners.

Copyright (c) 1999-2006 by Wingware. All rights reserved.:

```
Wingware
P.O. Box 1937
Brookline, MA 02446
United States of America
```

Introduction

Thanks for choosing Wingware's Wing IDE! This manual will help you get started and serves as a reference for the entire feature set of this product.

The manual is organized by major functional area of Wing IDE, including source code editor, source code editor, project manager, source browser (Wing IDE Professional only), and debugger.

Several appendices document the entire command set, provide pointers to resources and tips for Wing and Python users, and list the full software license.

The rest of this chapter describes how to install and start using Wing IDE. If you hate reading manuals, you should be able to get started by reading this chapter only.

> **Key Concepts**
> Throughout this manual, key concepts, important notes, and non-obvious features are highlighted in the same way as this paragraph. If you are skimming only, look for these marks.

1.1. Product Levels

This manual is for the Wing IDE Professional product level of the Wing IDE product line, which currently includes Wing IDE Personal and Wing IDE Professional.

Wing IDE Professional is the full-featured Wing IDE product, and may be licensed for commercial or non-commercial uses. Wing IDE Personal is for non-commercial use only and contains a subset of the features found in Wing IDE Professional. Both products are commercial products for sale from our website; Wing IDE Personal is not a free download.

Wing IDE Professional and Wing IDE Personal are independent products and may be installed at the same time on your system without interfering with each other.

For a list of features that have been omitted from Wing IDE Personal, please refer to http://wingware.com/wingide/features.

1.2. Licenses

Wing IDE is licensed per developer, and requires a separate license for each class of operating system (Windows, Linux, or OS X) that is used by the developer.

For the full license text, see the **Software License**.

License Activation

To run for more than 10 minutes, Wing IDE requires activation of a trial or permanent license. This system is designed to minimize the burden on legitimate user while eliminating casual license sharing that threatens our ability to support further development of Wing IDE.

An activation ties the license to the machine through a series of checks of the hardware connected to the system. This information is never transmitted over the internet, instead an SHA hash of some of the values is passed back and forth so that the machine will be identifiable without us knowing anything specific about it.

The machine identity metrics used for activation are designed to be forgiving so that replacing parts of your machine's hardware or upgrading the machine will usually not require another activation. By the same token, activating multiple times on the same machine (for example if the activation file is lost) usually does not increase your activation count.

Licenses come with three activations by default and additional activations can be obtained from the self-serve license manager or by emailing sales at wingware.com. As a fall-back in cases of emergency where we cannot be contacted and you don't have an activation, Wing IDE will run for 10 minutes at a time without any license at all.

See **Installing Your License** for more information on obtaining and activating licenses.

1.3. Supported Platforms

This version of Wing IDE is available for Microsoft Windows, Linux, Mac OS X, and some other operating systems for which contributed builds are available or where customers are willing to compile the product from source code.

Microsoft Windows

Wing IDE supports Windows 2000, XP, and 2003 Server. Windows 95, 98, and ME are not supported and will not work. Windows NT4 is not supported but *may* work with IE5+ installed.

Linux/Intel

Wing IDE runs on Linux versions with glibc2.2 or later (anything roughly 3 years old or newer; e.g. RedHat 7.1+, Mandrake 8.0+, SUSE 7.1+, and Debian 3.0+).

On Suse, you may need to install the gmp and python packages, or install Python from source, since Python is not installed by default here.

On Debian, you can convert the Wing IDE RPM into a Debian friendly package using the `alien` module. Install `alien`-Package, then `alien -d wingide-*.i386.rpm` followed by `dpkg -i wingide-*.deb`. Alternatively, use the Wing IDE tar file installer.

Mac OS X

Wing IDE runs on Mac OS X 10.3.9+. Wing IDE for OS X also requires an X11 Server and Window Manager. For details see **OS X Quick Start Guide**.

Only Python 2.2 and later are supported for Mac OS X. OS X 10.3 and later come with a standard version of Python already installed.

Other Platforms

Wing IDE can be compiled from source by customers wishing to use it on other operating systems (such as Linux PPC, Free BSD, or Solaris). This requires a non-disclosure agreement.

Some contributed builds of Wing IDE are available for other operating systems.

1.4. Supported Python versions

Before installing Wing, you must download and install Python 1.5.2, Python 2.0, Python 2.1, Python 2.2, Python 2.3, Python 2.4, or Python 2.5 if you do not already have one of these on your machine.

On Windows, Python must be installed using one of the installers from the python.org (or by building from source if desired).

On Linux, most distributions come with Python. Installing Python is usually only necessary on SUSE or a custom-built Linux installation.

On SUSE Linux, you can install the gmp and python packages that come with your distribution, or install from the materials available through the links given above.

On Mac OS X, Wing IDE only supports Python 2.2 and later.

1.5. Technical Support

If you have problems installing or using Wing IDE, please submit a bug report or feedback using the `Submit Bug Report` or `Submit Feedback` items in Wing IDE's `Help` menu.

Wingware Technical Support can also be contacted by email at support at wingware.com, or online at http://wingware.com/support.

Bug reports can also be sent by email to bugs at wingware.com. Please include your OS and product version number and details of the problem with each report.

If you are submitting a bug report via email, see **Obtaining Diagnostic Output** for more information on how to capture a log of Wing IDE and debug process internals. Whenever possible, these should be included with email-based bug reports.

1.6. Prerequisites for Installation

To run Wing IDE, you will need to obtain and install the following, if not already on your system:

Prerequisites for all platforms:

- Downloaded or CD version of Wing IDE
- A **supported version of Python**
- A working TCP/IP network configuration (for the debugger; no outside access to the internet is required)

Additional Prerequisities for Mac OS X:

- An X11 window server, such as Apple X11 for OS X (recommended) or XDarwin
- A window manager. Apple's X11 Server includes one; other options include OroborOSX

1.7. Installing

Before installing Wing IDE, be sure that you have installed the **necessary prerequisites**. If you are upgrading from a previous version, see **Upgrading** first.

Note: On all platforms, the installation location for Wing IDE is referred to as `WINGHOME`.

Windows 2000 and XP

Install Wing IDE by running the downloaded executable. Wing's files are installed by default in `C:\Program Files\Wing IDE`, but this location may be modified during installation. Wing will also create a **User Settings Directory** in the location appropriate for your version of Windows. This is used to store preferences and other settings.

Linux (glibc 2.2+)

To install the RPM, you must become root and type `rpm -i wingide-2.1.4-1.i386.rpm` The IDE will be installed in `/usr/lib/wingide2.1` with executables in `/usr/bin`. The Wing IDE executable is `/usr/bin/wing2.1`.

To install the tar file, type `tar -zxvf wingide-2.1.4-1-i386-linux.tar.gz` to extract from the tar archive. Then cd to the newly created directory `wingide-2.1.4-1-i386-linux`, type `./wing-install.py`, and answer the questions to specify where the program files and executable should go.

The Wing IDE executable is called `wing2.1`. You may have to modify your PATH environment if you've installed the executables in a directory that isn't already on your path. You may also have to open a new shell or request that your shell rescan the disk for executables (for example, with 'rehash' under tcsh).

Wing will create a **User Settings Directory** in `~/.wingide2`, which is used to store preferences and other settings.

For more information, see the **Linux installation details**.

Mac OS X 10.3+

Wing IDE on Mac OS X requires that you first install an X11 Server. For details on installing and running on OS X, see the **OS X Quickstart**.

1.8. Running the IDE

For a quick introduction to Wing's features, refer to the **Wing IDE Quickstart Guide**. For a more gentle in-depth start, see the **Wing IDE Tutorial**.

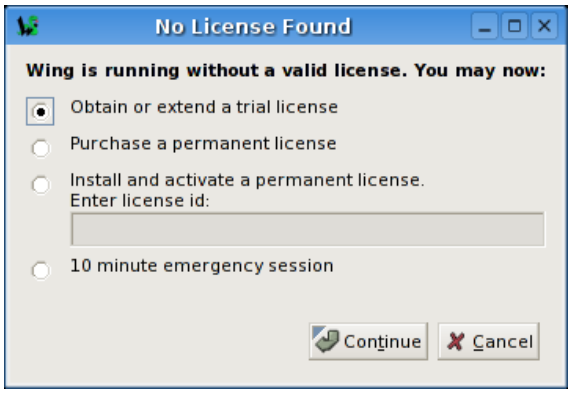
On Windows, start Wing IDE from the Program group of the Start menu. You can also start Wing from the command line with `wing.exe` (located inside `WINGHOME`).

On Linux/Unix, just execute `wing2.1` (or `wing` located inside `WINGHOME`)

On Mac OS X, first launch your X11 Server. Then start Wing IDE by double clicking on the app folder. If you launch Wing from the command line using `Contents/MacOS/wing` inside the Wing IDE app folder, then you may need to set your `DISPLAY` environment variable.

1.9. Installing your License

To run for more than 10 minutes, Wing IDE requires a trial or permanent license and the license needs to be activated (see the **Licenses** section for general information). When Wing IDE is first started, you can obtain a trial licence, purchase a permanent license, install & activate a permanent license, or use Wing for up to 10 minutes without a license:


No License Found
Wing is running without a valid license. You may now:
Obtain or extend a trial license
Purchase a permanent license
Install and activate a permanent license.
Enter license id:
10 minute emergency session
Continue
Cancel

Trial Licenses

Trial licenses allow evaluation of Wing IDE for 10 days, with an option to extend the evaluation twice for up to 30 days total. The most convenient way to obtain a trial license is to ask Wing IDE to connect directly to TCP port 80 (http) on `wingware.com`. In this case, Wing will attempt to connect and get everything it needs to run for 10 days.

After the trial license is obtained, Wing will not attempt to connect to `wingware.com` (or any other site) unless you submit feedback or a bug report through the `Help` menu.

Activate Trial License
Connect Wing IDE directly to wingware.com
Activate manually at http://wingware.com/activate. You will need your license id TNX3P-BFECQ-CYW59-6QF2T and request code RLX24-L6WNK-DG88X-HM7WL. Then enter the provided activation key here:
Continue
Cancel

If you're unable or unwilling to connect Wing IDE directly to `wingware.com`, you can go to http://wingware.com/activate and enter the license id and activation request number. After entering this information, you will be given an activation key which you can enter into Wing's dialog box to complete the activation. This is exactly the same exchange of information that occurs when Wing IDE connects directly to `wingware.com` to obtain a trial license.

If you run into problems or need additional evaluation time, please email us at sales at wingware.com.

Permanent Licenses

Permanent licenses and upgrades may be purchased in the online store at http://wingware.com/store. Permanent licenses include free upgrades through the 2.* version series. Wing IDE Professional licenses also allow access to the product source code via http://wingware.com/downloads (requires signed non-disclosure agreement).

Activating on Shared Drives

When Wing is installed on a shared drive (for example a USB keydrive, or on a file server), the **User Settings Directory** where the license activation is stored may be accessed from several different computers.

In this case, Wing must be activated once on each computer. The resulting extra activations will be stored as `license.act1`, `license.act2`, and so forth, and Wing will automatically select the appropriate activation depending on where it is running.

Obtaining Additional Activations

If you run out of activations, you can use the self-serve license manager or email us at sales at wingware.com to obtain additional activations on any legitimately purchased license.

1.10. User Settings Directory

The first time you run Wing, it will create your `User Settings Directory` automatically. This directory is used to store your license, preferences, auto-save files, recent lists, and other files used internally by Wing. If the directory cannot be created, Wing will exit.

The settings directory is created in a location appropriate to your operating system. The location is listed as your `Settings Directory` in the **About Box** accessible from the `Help` menu.

These are the locations used by Wing:

Linux/Unix -- `~/.wingide2` (a sub-directory of your home directory)

Windows -- In `Wing IDE 2` within the `Application Data` folder. The location varies by version of Windows. For Windows 2000 and XP running on `c:` with an English localization the location is:

```
c:\\Documents and Settings\\${username}\\Application Data
```

Wing also creates a Cache Directory that contains the source analysis cache. This is often but not always in the same location as the above. On Windows, this directory is usually in the corresponding Local Settings area under Documents and Settings. This directory is also listed in the **About Box**.

1.11. Upgrading

If you are upgrading Wing and have previously installed patch files by hand (rather than with the integrated patch manager), see this **additional information** to avoid problems during your upgrade.

If you are upgrading within the same minor version number of Wing (for example from 2.1 to 2.1.3) this will replace your previous installation. Once you have upgraded, your previous preferences and settings should remain and you should immediately be able to start using Wing.

If you are upgrading across major releases (for example from 2.0 to 2.1), this will install the new version along side your old version of Wing.

To install an upgrade, follow the steps described in **Installing**

1.11.1. Fixing a Failed Upgrade

In rare cases when upgrading within minor versions (for example 2.1 to 2.1.1), the upgrade may fail to overwrite old files, resulting in random or bizarre behaviors and crashing. The fix for this problem is to completely uninstall and manually remove remaining files before installing the upgrade again.

Windows

To uninstall on Windows, run the Add/Remove Programs control panel to uninstall Wing IDE. Then go into the directory where Wing was located and manually remove any remaining folders and files.

Linux RPM

If you installed Wing IDE for Linux from RPM, issue the command `rpm -e wingide`. Then go into `/usr/lib/wingide` and remove any remaining files and directories.

Linux Tar

If you installed Wing IDE for Linux from the tar distribution, find your Wing installation directory and run the `wing-uninstall` script located there. Once done, manually remove any remaining files and directories.

Mac OS X

On Mac OS X, you are simply copying from a disk image to create the installation. Always remove your old installation first.

1.12. Advanced Installation

This section describes installation options for advanced users.

1.12.1. Installing Extra Documentation

If you are using Linux/Unix, the Python manual is not included in most installations, so you may also wish to download and install local copies of these pages.

Place the top-level of the HTML formatted Python manual (where `index.html` is found) into `python-manual/#.#` within your Wing IDE installation. Substitute for `#.#` the major and minor version of the corresponding Python interpreter (for example, for the Python 2.3.x manual, use `python-manual/2.3`).

Once this is done, Wing will use the local disk copy rather than going to the web when the Python Manual item is selected from the Help menu.

1.12.2. Linux Installation Notes

On Linux, Wing can be installed from RPM or from tar archive. Use the latter if you do not have root access on your machine or wish to install Wing somewhere other than `/usr/lib/wingide`.

Installing from RPM:

Wing can be installed from an RPM package on RPM-based systems, such as RedHat and Mandriva. To install, run `rpm -i wingide-2.1.4-1.i386.rpm` as root or use your favorite RPM administration tool to install the RPM. Most files for Wing are placed under the `/usr/lib/wingide` directory and a link for the `wing2.1` command is placed in the `/usr/bin` directory.

The installation location of Wing is referred to as `WINGHOME`. If you installed from RPM, this will always be `/usr/lib/wingide`.

Installing from Tar Archive:

Wing may also be installed from a tar archive. This can be used on systems that do not use RPM, or if you wish to install Wing into a directory other than `/usr/lib/wingide`. Unpacking this archive with `tar -zxvf wingide-2.1.4-1-i386-linux.tar.gz` will create a `wingide-2.1.4-1-i386-linux` directory that contains the `wing-install.py` script and a `binary-package.tar` file.

Running the `wing-install.py` script will prompt for the location to install support files for Wing (`WINGHOME`), and the location in which to create a symbolic link to `wing2.1`. These locations default to `/usr/local/lib/wingide` and `/usr/local/bin`, respectively. The install program must have read/write access to both of these directories, and all users running Wing must have read access to both.

The installation location of Wing is referred to as `WINGHOME`. If you installed from tar, this will be the location you chose when you ran the installer.

Installing on Debian Linux:

You can convert the Linux RPM into a Debian friendly package using the `alien` module. Here's how:

1) Download the RPM package.

2) Install the alien-Package, which is part of the Debian Package Collection. Use apt-get, kpackage, aptitude, or other package manager to find and install it.

3) Change into the directory where the Wing IDE RPM is located and issue the following command at the cmd prompt:

```
alien -d wingide-2.1.4-1.i386.rpm
```

4) A `wingide-2.1.4-1.deb` file can now be found in the same directory. To install the this package issue the following command at the prompt:

```
dpkg -i wingide-2.1.4-1
```

This is a general technique that also works for other Linux RPMs.

Using System-wide GTK:

By default, Wing IDE runs with its own copy of GTK2 and does not pick up on the system-configured theme. This is done to avoid problems and bugs sometimes brought out by binary incompatibilities in GTK versions.

On Linux versions that include GTK version 2.6 or later, you can tell Wing IDE to use the system-defined GTK2 by setting the **System GTK** preference or running with the `--system-gtk` command line argument.

Using the system-wide GTK2 in this way generally works quite well but may result in crashing or display bugs due to binary incompatibilities in GTK and related libraries. If you set the preference and Wing fails to start, specify the `--private-gtk` command line option to override the preference.

Non-ascii File Paths on Older Linux Systems:

Some older Linux versions require setting the environment variable G_BROKEN_FILENAMES before Wing IDE's file open/save dialog will work properly with file paths that contain non-ascii characters. The environment variable is already set on some systems where it is needed but this is not always the case.

1.12.3. Source Code Installation

Source code is available to licensed users of Wing IDE Professional (non-evaluation licenses only) who have completed a non-disclosure agreement. Upon receipt of this agreement, you will be provided with instructions for obtaining and working with the product source code.

1.13. Removing Wing IDE

Windows

On Windows, use the Add/Remove Programs control panel, select Wing IDE and remove it.

Linux/Unix

To remove an RPM installation on Linux, type `rpm -e wingide`.

To remove a tar archive installation on Linux/Unix, invoke the `wing-uninstall` script in `WINGHOME`. This will automatically remove all files that appear not to have been changed since installation, It will ask whether it should remove any files that appear to be changed.

Mac OS X

To remove Wing from Mac OS X, just drag its application folder to the trash.

Removing a Patched Installation

If you previously applied patch files to your Wing installation, you may need to manually remove remaining files and directories after uninstallation. In this case you should retain the **User Settings Directory**, which contains license and preferences information.

1.14. Command Line Usage

Whenever you run `wing2.1` from the command line, you may specify a list of files to open. These can be arbitrary text files and a project file. For example, the following will open project file `myproject.wpr` and also the three source files `mysource.py`, `README`, and `Makefile`:

```
wing2.1 mysource.py README Makefile myproject.wpr
```

(on Windows, the executable is called `wing.exe`)

Wing determines file type by extension, so position of the project file name (if any) on the command line is not important.

The following valid options may be specified anywhere on the command line:

--prefs-file -- Add the file name following this argument to the list of preferences files that are opened by the IDE. These files are opened after the system-wide and default user preferences files, so values in them override those given in other preferences files.

--new -- By default Wing will reuse an existing running instance of Wing IDE to open files specified on the command line. This option turns off this behavior and forces creation of a new instance of Wing IDE. Note that a new instance is always created if no files are given on the command line.

--reuse -- Force Wing to reuse an existing running instance of Wing IDE even if there are no file names given on the command line. This just brings Wing to the front.

--system-gtk -- *(Posix only)* This option causes Wing to try to use the system-wide install of GTK2 rather than its own version of GTK, regardless of any preference setting. Running in this mode will cause Wing to pick up on system-wide theme defaults, but may result in crashing or display problems due to incompatibilities in GTK and related libraries.

--private-gtk -- *(Posix only)* This option causes Wing to use it's private copy of GTK2 and related libraries, regardless of any preference settings. Use of private GTK may result in Wing not matching the system-wide theme, but also will avoid incompatibilities with the system-wide GTK library.

--verbose -- *(Posix only)* This option causes Wing to print verbose error reporting output to `stderr`. On Windows, run `console_wing.exe` instead for the same result.

--display -- *(Posix only)* Sets the X Windows display for Wing to run with. The display specification should follow this argument, in standard format, e.g. `myhost:0.0`.

--use-winghome -- *(For developers only)* This option sets WINGHOME to be used during this run. It is used internally and by developers contributing to Wing IDE. The directory to use follows this argument.

--use-src -- *(For developers only)* This option is used to force Wing to run from Python source files even if compiled files are present in the `bin` directory, as is the case after a distribution has been built.

--orig-python-path -- *(For developers only)* This option is used internally to indicate the original Python path in use by the user before Wing was launched. The path follows this argument.

--squelch-output -- *(For developers only)* This option prevents any output of any kind to `stdout` and `stderr`. Used on Windows to avoid console creation.

1.15. Trouble-shooting

This section describes what to do if you are having trouble installing or using Wing IDE.

> We welcome feedback and bug reports, both of which can be submitted directly from Wing IDE using the `Submit Feedback` and `Submit Bug Report` items in the Help menu, or by emailing us at support at wingware.com.

1.15.1. Trouble-shooting Failure to Start

If you are having trouble getting Wing to start at all, read through this section for information on diagnosing the problem.

On OS X, Wing requires that you install and launch an X11 Server before starting Wing IDE. See the **OS X How-To** for details.

On Windows, the user's temporary directory sometimes becomes full, which prevents Wing from starting. Check whether the directory contains more than 65,534 files. Some versions of Acrobat Reader will leave large numbers of lock files in this directory. These files are named `Acrxxxx.tmp`.

On Fedora Core 5 and other Linuxes with SELinux, Wing won't start because permissions are denied on one of the shared libraries needed by it. The solution is to change into `bin/2.4/external/pyscintilla2` and issue the following command:

```
chcon -t texrel_shlib_t _scintilla.so
```

On Linux, in some cases, Wing will not run with its own private GTK installation because of incompatibilities with the system. To test this, run Wing with the `--system-gtk` command line option after making sure your Linux system has the GTK packages installed. If this works, you can set the **Use System Gtk** preference.

Note, however, that there are known problems running system-provided Qt emulation when using the system GTK option. Some of these themes contain bugs that can cause crashing. If you need to use the system GTK and experience crashes, we recommend using a theme other than a Qt theme.

On Linux, if Wing fails to start after the **Use System Gtk** preference has been set, use the `--private-gtk` command line option to get Wing running again so that the preference can be turned off.

To rule out problems with a project file or preferences, try renaming your **User Settings Directory** and restart Wing. If this works, you can copy over files from the

renamed directory one at a time to isolate the problem -- or email support at wingware dot com for help.

Under a Windows terminal server, Wing may not be able to set up the environment variables it uses internally and will not start up. In this case, you can get Wing to start with the following commands:

```
set PYTHONOPTIMIZE=1
set PYTHONHOME=D:\Program Files\WingIDE\bin\PyCore
wing.exe
```

Alter `PYTHONHOME` according to the location at which you've installed Wing IDE.

In other cases, refer to **Obtaining Diagnostic Output**.

1.15.2. Issues on Microsoft Windows

Wing has a few problems and limitations on Microsoft Windows systems

1) A few of the demo shell extension COM objects from win32all can cause Wing to crash if they are registered. The crash occurs when the file open, save, and add files to project dialog boxes are used. These extensions may be disabled by using ShellExView (http://www.snapfiles.com/get/shellexview.html) or a similiar program to find and disable them. They can also be uninstalled by running the .py file with a --unregister argument.

2) The nVidia Desktop Manager may cause the system to freeze on some versions of Windows (apparently the card becomes very sluggish while the system CPU utilization remains near 0%). The problem appears more frequently when using Wing in multi-window modes but may occur in all cases. Disabling the manager prevents the freeze from occuring.

There may be other display issues (such as failure to draw window contents when un-minimizing from Windows task bar) specifically with some nVidia cards, even if the desktop manager is disabled. We are investigating and plan to fix or work around this problem in the future.

3) Windows drag-n-drop currently doesn't work for transferring data (text or files) between Wing and either the Windows desktop or other applications.

1.15.3. Trouble-shooting Failure to Debug

If you have trouble debugging with Wing IDE, select which of the following most closely describes the problem you are seeing:

- **Debugging fails to start**
- **Debugger reports exceptions not seen outside Wing**
- **Debugger doesn't stop on breakpoints**
- **Debugger doesn't stop on exceptions**

1.15.3.1. Failure to Start Debug

Wing may fail to start the debug process in certain cases. If this happens, it often helps to try debugging a small test such as the following:

```
print "test1"
print "test2"
```

Use the `Start / Continue` command from the Debug menu to cause Wing IDE to attempt to run only as far as the first line of your code. This rules out possible problems caused by specific code.

Then check through the following common problems. For information on obtaining additional information from the debug sub-system, refer to the **Diagnostic Output** section:

1) Wing's debugger uses a TCP/IP protocol to communicate with the IDE. Make sure that TCP/IP is installed and configured on your machine. If you are running a custom-built copy of Python, verify that the `socket` module is available.

2) If Wing says it can't find Python or if you've got multiple versions of Python on your system, make sure you've got your **Project Properties** set up to contain a valid interpreter (see Source / Show Analysis Stats menu item to verify that the right interpreter is being found).

3) Enter any necessary `PYTHONPATH` for your debug process in Project Properties if not already defined in the environment.

4) If you set `PYTHONHOME` or `PYTHONPATH` environment variables, these may cause the debug process to fail if they do not match the particular Python interpreter that Wing is launching. You can either change the interpreter used so it matches, or unset or alter

these environment values from the outside or via Project Properties from the Project menu.

`PYTHONHOME` is a problem in all cases when it doesn't match the Python interpreter reported in the Source menu's Show Analysis Stats dialog.

`PYTHONPATH` is only a problem if it contains directories that are part of a Python installation. When this doesn't match the interpreter version, this leads to import errors because Python tries to import incompatible modules.

5) On Windows, check that you don't have Hummingbird Socks Client installed on your machine. Some versions and configurations of this product are known to incorrectly route network packets in such a way that slows down the Wing IDE debugger enough to make it time out during initialization.

6) All forms of the Python binary distribution (TAR, RPM, and Windows installer) are known to have problems when a newer version of Python is installed directly over an older one on disk.

In this case, most Python programs will appear to work fine outside of Wing IDE but will not work within the Wing IDE debugger. This occurs because the debug support code uses sockets and other functionality that is not necessarily exercised by your debug program outside of the Wing debugger.

If you try to run a debug session in Wing IDE and it fails, you may be having this problem. The following test script can be used to confirm that the problem exists in your Python installation (although it is not known whether this script shows exceptions in all cases where a corrupt Python installation is the cause of debugging problems):

```
import sys
print 'sys.version =', sys.version
print 'sys.executable =', sys.executable
print 'sys.version_info =', sys.version_info
import socket
print 'socket =', socket
print 'socket._socket =', socket._socket
import select
print 'select =', select
import cPickle
print 'cPickle =', cPickle
```

To solve this problem, try uninstalling Python, manually removing any remaining files, and installing again. Or install Python into a new location on disk.

Once this is done, be sure to confirm that Wing is configured to use the new Python installation from the Project Properties dialog in the Project menu and that the Show Analysis Stats item in the Source menu displays the correct intepreter.

7) Wing's debugger is unable to debug games written with pygame when they are running in full screen mode. Use window mode instead. This is a problem also for other IDEs; we have not yet investigated the cause.

1.15.3.2. Extra Debugger Exceptions

Wing's debugger sometimes appears to reveal bugs that are not seen when running outside of the debugger. This is a result of how Wing decides which exceptions should be shown to the user -- it is inspecting exceptions as they are raised and making decisions about whether or not the exception is unexpected or part of normal operation.

You can train Wing to ignore unwanted exception reports with the checkbox in the `Exceptions` tool.

You can also change the way Wing reports debug process exceptions with the **Exception Reporting** preference.

For more information, see **Managing Exceptions**.

1.15.3.3. Failure to Stop on Exceptions

By default, Wing only stops on exceptions it thinks will be unhandled. If your code runs within a catch-all try/except clause written in Python (as in some GUI main loops or in an environment like Zope), Wing will not report any exceptions encountered in your debug process, except if that exception leads the debug process to exit.

To get Wing to stop, you can set the **Exception Reporting** preference to `All Immediately`. However, this often reports many other exceptions that happen internally during normal operation.

An alternative is to recode your app to make your catch-all exception handler optional as in the following example:

```
import os

# No handler when running in Wing's debugger
if os.environ.has_key['WINGDB_ACTIVE']:
  dosomething()
```

```
# Handle unexpected exceptions gracefully at other times
else:
  try:
    dosomething()
  except:
    # handler here
```

Alternatively, add the following to your catch-all exception handler:

```
import os
if os.environ.has_key['WINGDB_ACTIVE']:
  raise
```

This will terminate your debug process on the exception but will not always allow you to inspect program state at the time that the exception is raised.

Note that environments such as wxPython, PyGTK, and others include catch-all handlers for unexpected exceptions raised in the main loop, but those handlers are written in C/C++ extension module code and thus will be reported correctly by Wing without any modification to the handler.

1.15.3.4. Failure to Stop on Breakpoints or Show Source Code

The most common cause of failure to stop on breakpoints or to bring up source windows while stopping or stepping through code is a mismatch between the file name that is stored in the `*.pyc` file and the actual location of the `*.py` source file.

This can be caused by (1) not saving before you run in the debugger, (2) using partial path names on PYTHONPATH or when invoking a script from the command line (the partial path stored in the `*.pyc` file may become invalid if current directory changes), (3) moving around the `*.pyc` file after they are created, or (4) using `compileall.py` to create `*.pyc` files from source. The easiest way to solve this is to use only full paths on PYTHONPATH and remove any suspect `*.pyc` files.

Wing may fail to stop when debugging a multi-threaded application, because the debugger can currently only debug one thread at a time. If a breakpoint is reached in a thread other than the main thread (or, if debugging an externally launched process, the thread on which `wingdbstub` was imported), then the breakpoint is ignored.

Less common causes of this problem are (1) running Python with the `-O` optimization option, (2) running Python with `psyco` or other optimizer, (3) overriding the Python `__import__` routine, (4) adding breakpoints after you've started debugging an application that spends much of its time in C/C++ or other non-Python code, and (5) on win32,

using symbolic links to directories that contain your source code files (Posix platforms handle symbolic links just fine).

For more information, see the **Debugger Limitations** section.

1.15.4. Obtaining Diagnostic Output

Wing IDE and your debug code run in separate processes, each of which can independently be configured to collect additional diagnostic log information.

Diagnosing General IDE Problems

A quick way to diagnose problems seen while working with Wing IDE is to submit a bug report from the `Help` menu. Please include a description of the problem and check the `Include error log` checkbox so we can diagnose and fix the problem.

To diagnose other problems, such as failure to start, try looking at the file `error-log` in your **User Settings Directory**.

Alternatively, run `console_wing.exe` (on Windows) or `wing --verbose` (on Linux/Unix and OS X) from the command line to display diagnostic output.

Email this output to support at wingware.com along with your system version, version of Wing IDE, and any other potentially relevant details.

Diagnosing Debugger Problems

To diagnose debugger problems, set preference **Debug Internals Log File** to a value other than `No logging` and turn on preferences **Use External Console** and **External Console Waits on Exit**. When you try again, Wing will display a debug console with diagnostics.

Alternatively, copy `wingdbstub.py` out of your Wing IDE installation, set `WINGDB_LOGFILE` environment variable to `<stderr>` or the name of a log file on disk (or alter `kLogFile` inside `wingdbstub.py`), set the **Enable Passive Listen** preference to True, and try launching the following script from the command line:

```
import wingdbstub
print "test1"
print "test2"
```

This prints diagnostic output that may be easier to capture in some cases.

Email this output to support at wingware.com. Include also the contents of the file

`error-log` in your **User Settings Directory** , and also your system version, version of Wing IDE, and any other potentially relevant details.

1.15.5. Speeding up Wing

Wing should present a responsive, snappy user interface even on relatively slow hardware. In some cases, Wing may appear sluggish:

1) Try using a different Display Theme from preferences -- the pixmap manipulations in Wing's default themes sometimes fail to be accelerated on certain display hardware. This is a bug we are working on fixing; it seems to appear on faster hardware more than on slower hardware.

2) If you have nVidia desktop manager, disable it for Wing.

3) The first time you set up a project file, Wing analyses all source files for the source code browser and auto-completion facilities. During this time, the browser's class-oriented views will display only the source constructs from files of which analysis information has already been obtained. The user interface may also appear to be sluggish and Wing will consume substantial amounts of CPU time.

To limit this effect in subsequent sessions, Wing stores its source analysis information to disk in a cache within your **User Settings Directory**.

However, with large projects even reading this cache and checking files for updates may take a while when Wing is first started. This process happens in the background after launch and takes 7-15 seconds per 100,000 lines of code on a Celeron 400 processor (almost not noticable on any modern hardware).

In all cases, Wing will eventually complete this process and should at that time consume almost no CPU during normal editing and debugging.

4) In wxPython and other code that uses `from xxx import *` style imports, the auto-completer may initially be slow to appear if it needs to process many hundreds of symbols. This should only happen the first time it appears, however.

5) On Windows, if Wing is started while operating via Remote Desktop Connection, performance is terrible, even after quitting the RDC session and working directly on the machine that is running Wing. However, if Wing is started on the machine on which it runs, performance is very lively on that machine and acceptable if switched to operating via RDC without quitting Wing.

6) Some users have reported Hummingbird Socks Client for Windows to cause the debugger to slow down substantially, apparently as a result of improperly routed TCP/IP packets.

7) If you are displaying Wing remotely via X11, try turning off anti-aliased fonts by placing this file in `~/.fonts.conf` on the display machine and then restarting the X server.

1.15.6. Trouble-shooting Failure to Open Filenames Containing Spaces

On Windows: When using Windows File Types or Open With to cause Python files to be opened with Wing, some versions of Windows set up the wrong command line for opening the file. You can fix this using *regedt32.exe*, *regedit.exe*, or similar tool to edit the following registry location:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Applications\wing.exe\shell\open\command
```

The problem is that the association stored there is missing quotes around the *%1* argument. It should instead read as follows:

```
"C:\Program Files\Wing IDE\bin\wing.exe" "%1" %*
```

On Linux: KDE's Konqueror has the same problem that file names passed on the command line to applications bound to a file type are not enclosed with quotes, so the command line is not parsed correctly. We do not currently have a work-around for this problem.

1.15.7. Trouble-shooting Failure to Print

This section provides some hints to get printing working if it doesn't work "out of the box".

On Windows

Wing has trouble printing with some printer drivers. One known issue is failure to transfer the correct font to the printer. The symptom is correctly printed header and footer but gibberish in the body of the source code. The problem can be solved in the Advanced menu under Print Properties in Windows by changing TrueTypeFont from "substitute with device font" to "download as soft font".

On Linux

For Python files, Wing prints PDF formatted output directly to the printer. This does not work on at least some Linux distributions and can be worked around by setting the **Print Spool Command** preference to `pdf2ps %s - | kprinter --stdin`.

Wing uses `kprinter` by default on Linux when it is present. Another problem on Linux occurs when using a buggy version of `kprinter`, To rule that out, try `pdf2ps %s - | lpr` or simply `lpr %s` instead for the **Print Spool Command** preference

Turning on the **Print Python as Text** preference may also solve some printing problems, although on some systems with plainer output for Python files. When this is enabled, Python files are also passed through the the command given in the **Text Print Cmd** preference instead of generating syntax highlighted PDF. In all cases, all non-Python files are passed through this command.

Customization

There are many ways to customize Wing IDE in order to adapt it to your needs or preferences. This chapter describes the options that are available to you.

These are some of the areas of customization that are available:

- The content, layout, and look of the IDE windows can be configured
- Many other options are available through preferences
- The editor can run with different personalities such as Vim, Emacs, Visual Studio, and Brief emulation
- Keyboard shortcuts can be added, removed, or altered for any Wing command
- File sets can be defined to control some of the IDE features

2.1. User Interface Options

Wing provides many options for customizing the user interface to your needs. Preferences can be set to control the number and style of windows used when working with the IDE, the layout of tools within windows, display text font, size, and color, the style and content of the toolbar, and the overall look or "theme" (including white on black and many others).

2.1.1. Windowing Policies

Wing IDE can run in a variety of windowing modes. This is controlled by the **Windowing Policy** preference, which provides the following options:

- **Combined Tool Box and Editor Windows** -- This is the default, in which Wing opens a single window that combines the editor area with two tool box panels.
- **Separate Tool Box Windows** -- In this mode, Wing IDE moves all the tools out to a separate shared window.
- **One Window Per Editor** -- In this mode, Wing IDE creates one top-level window for each editor that is opened. Additionally, all tools are moved out to a separate shared tool box window and the toolbar and menu are moved out to a shared toolbar/menu window.

The windowing policy is used to describe the initial configuration and basic action of windows in the IDE. When it is changed, Wing will reconfigure your projects to match the windowing policy the first time they are used with the new setting.

However, it is possible to create additional IDE windows and to move editors and tools out to another window or among existing windows without changing from the default windowing policy. This is described below.

2.1.2. User Interface Layout

When working in the default windowing policy, Wing's main user interface area consists of two tool boxes (by default at bottom and right, but this can be altered in **Preferences**) and an area for source editors and integrated help.

Clicking on an already-active notebook tab will cause Wing to minimize the entire panel so that only the notebook tabs are visible. Clicking again will return the tool box to its former size. The F1 and F2 keys toggle between these modes. The command `Maximize Editor Area` in the `Tools` menu (Shift-F2) can also be used to quickly hide both tool areas and toolbar.

In other windowing modes, the tool boxes and editor area are presented in separate windows but share many of the configuration options described below.

Configuring the Toolbar

Wing's toolbar can be configured by altering the size and style of the toolbar icons in the toolbar, and whether or not text is shown in addition to or instead of icons. This is controlled with the **Toolbar Icon Size** and **Toolbar Icon Style** preferences.

Alternatively, the toolbar can be hidden completely with the **Show Toolbar** preference.

Configuring the Editor Area

The options popup in the top right of the editor area allows for splitting and joining the editor into multiple independent panels. These can be arranged horizontally, vertically, or any combination thereof. When multiple splits are shown, all the open files within the window are available within each split, allowing work on any combination of files and/or different parts of the same file.

The options popup menu can also be used to change between tabbed editors and editors that show a popup menu for selecting among files (the latter can be easier to manage with large number of files) and to move editors out to a separate window or among existing windows when multiple windows are open.

Configuring Tool Boxes

Each of the tool boxes can be also be split or joined into any number of sub-panels along the axis of the notebook by clicking on the options drop down icon in the tab area of the notebooks (right-clicking also works). The number of tool box splits Wing shows by default depends on your monitor size.

The options pop down menu can also be used to duplicate tools, or move them around among the splits or out to separate windows.

The size of each panel and the panel splits can also be altered by dragging on the dividers between them.

All available tools are enumerated in the Tools menu, which will display the most recently used tool of that type or will add one to your window at its default location, if none is already present.

Creating Additional Windows

In addition to moving existing editors or tools to new windows, it is also possible to create new tool windows (initially with a single tool) and new document windows (with editor and toolbars if applicable to the selected windowing policy) from the Windows menu.

Wing IDE will remember the state of all windows as part of your project file, so the same window layout and contents will be restored in subsequent work sessions.

2.1.3. Altering Text Display

Wing tries to find display fonts appropriate for each system on which it runs, but many users will want to customize the font style and size used in the editor and other user interface areas. This can be done with the **Source Code Font/Size** and **Display Font/Size** preferences.

The color of text for some file types in the editor can be controlled with the **Syntax Formatting** preference.

Note that the when the **Source Code Background** preference is set to a color other than white, Wing will compute appropriately visible colors for text according to the chosen background color.

The color used for text selection can also be controlled with the **Text Selection Color** preference.

Changes in color preferences will often depend on the overall display theme that is chosen, as described in the next section.

2.1.4. Setting Overall Display Theme

Wing is based on GTK2, a cross-platform user interface toolkit that provides customizable `themes`, which control the overall look and feel of the user interface. Wing's default theme varies by platform (a Windows emulation theme is used on Windows, and an OS X like theme on OS X) and can be changed with the **Display Theme** preference.

In most cases, the new theme will instantly be applied to Wing's user interface. When switching back to default settings, a restart may be needed in some cases, as indicated by message dialog.

Some systems with slower graphics cards may not run as well using the more colorful 3D rendered themes. In this case, using the `Gtk Default` theme is the best option, as it involves no extra graphics-level processing.

System GTK on Linux

On Linux systems with GTK 2.6 or later installed, it is possible to run Wing with the system-wide GTK installation and system-defined themes. This is controlled with the **Use System GTK** preference or the `--system-gtk` or `--private-gtk` **command line arguments**. Wing works reasonably well with most 2.6.x GTK2 releases, but there still may be problems resulting from version differences. If you have any problems with Wing's stability or are seeing display glitches, you should use the private gtk option.

2.2. Preferences

Wing has many preferences that control features of the editor, debugger, source browser, project manager, and other tools.

To alter these, use the `Preferences` item in the `Edit` menu. This organizes all available

preferences by category and provides access to documentation in tooltips that are displayed when mousing over the label area to the left of each preference. Any non-default values that are selected through the **Preferences Dialog** are stored in the user's preferences file, which is located in the **User Settings Directory**.

2.2.1. Preferences File Layers

Wing's preferences manager runs on a layered set of preferences files. An installation-wide preferences file may be placed inside `WINGHOME` and individual users can override these values by placing a preferences file in their **User Settings Directory** (or using the Preferences GUI in Wing). The values given in the user-specific preferences file take precedence over any values in the default `WINGHOME/preferences` file.

It is also possible to specify additional preferences files on the command line though the `--prefs-file` option. For example:

```
wing2.1 --prefs-file /path/to/myprefs
```

Any file specified in this way will override values stored in the per-user or installation-wide preferences files.

2.2.2. Preferences File Format

While we recommend using the graphical preferences manager to alter preferences, some users may wish to edit the underlying text files directly.

The preferences file format consists of a series of sections separated by bracketed headers. Currently the only valid section is `[user-preferences]` and all other sections are ignored. The body of each section is a sequence of lines, each of which is a `name=value` pair.

Each preference name is in *domain.preference* form, where *domain* is the IDE subsystem affected and *preference* is the name of the specific preference (for example, `edit.personality` defines the source editor's runtime personality).

Preference values can be any Python expression that will evaluate to a number, string, tuple, list, or dictionary. Additionally, the constants `true` and `false` are defined and supported as valid values. Long lines may be continued by placing a backslash (\\) at the end of a line and comments may be placed anywhere on a line by starting them with `#`.

If you wish to write preferences files by hand, refer to the **Preferences Reference** for documentation of all available preferences.

2.3. Editor Personalities

The default editor personality for Wing implements most common keyboard equivalents found in a simple graphical text editor. This uses primarily the graphical user interface for interacting with the editor and limits use of complex keyboard-driven command interaction.

> **Emulation of Other Editors**
> The first thing any Vim, Emacs, Visual Studio, or Brief user will want to do is to set the editor personality to emulate their editor of choice. This is done with the **Keyboard / Personality** user interface preference.

Under the Vim and Emacs personalities, key strokes can be used to control most of the editor's functionality, using a textual interaction 'mini-buffer' at the bottom of the editor window where the current line number and other informational messages are normally displayed.

It is also possible to add, alter, or remove individual keyboard command mappings within each of these personalities. See **Key Equivalents** for details.

2.4. Key Equivalents

The command a key will invoke may be modified be selecting a different key map or by specifying a custom key binding. A custom key binding will override any binding for a particular key found in the keymap. Custom key bindings are set via the **Custom Key Bindings** preference.

To add a binding, click the insert button and then press the key to be bound in the `Key` field and enter the name of the command to invoke in the `Command` field.

Key bindings defined by default or overriden by this preference will be shown in any menu items that implement the same command. In cases where a command is given more than one key equivalent, only the last equivalent found will be displayed (although both bindings will work from the keyboard).

Key maps

Wing ships with several key equivalency maps found in `WINGHOME`, including `keymap.normal`, `keymap.emacs`, `keymap.vi`, among others. These are used as default key maps for the corresponding editor personalities.

For developing entirely new key bindings or debugging key bindings, it is possible to create a custom key equivalency map and use it as your default map through **Key Map File** preference. This is not recommended for casual users, since completely replacing the default key maps will require manual tracking of changes in commands across Wing versions.

In a key map file, each key equivalent is built from names listed in the **Key Names** section. These names can be combined as follows:

1) A single unmodified key is specified by its name alone, for example `'Down'` for the down arrow key.
2) Modified keys are specified by hyphenating the key names, for example `'shift-Down'` for the down arrow key pushed while shift is held down. Multiple modifiers may also be specified, as in `'ctrl-shift-Down'`.
3) Multi-key combinations can be specified by listing multiple key names separated by a space. For example, to define a key equivalent that consists of first pushing **ctrl-x** and then pushing the **a** key by itself, use `'ctrl-x a'` as the key sequence.

The command portion of the key equivalency definition may be any of the commands listed in section **Command Reference**. Use None to remove the given key equivalent entirely.

Specifying a key binding that already exists in the default key binding simply replaces that binding with your override.

Examples

Here is an example of adding a key binding for a command. If the command already has a default key binding, both bindings will work:

```
'Ctrl-X P': 'debug-attach'
```

This example removes a key equivalent entirely:

```
'Ctrl-C Ctrl-C': None
```

These can be combined to changes the key binding for a command without retaining its default key binding:

```
'Ctrl-C Ctrl-C': None
'Ctrl-G': 'debug-continue'
```

Wing always retains only the last key binding for a given key combination. This example binds Ctrl-X to 'quit' and no other command:

```
'Ctrl-X': 'debug-stop'
'Ctrl-X': 'quit'
```

2.4.1. Key Names

(1) Key modifiers supported by Wing IDE for key bindings are:

- **Ctrl** -- Either control key.
- **Shift** -- Either shift key. This modifier is ignored with some key names, as indicated below.
- **Alt** -- Not recommended for general use since these bindings tend to conflict with menu accelerators and operating system or window manager operations.
- **Command** -- Macintosh command / apple key. This may be mapped to other keys on other systems, but is intended for use on the Macintosh.

On Linux and OS X it is possible to remap the function of the Control, Alt, command, and windows keys. In those cases, the Ctrl and Alt modifiers will refer to the keys specified in that mapping.

(2) The digit keys and core western alphabet keys are specified as follows:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z,

(3) Most punctuation can be specified but any Shift modifier will be ignored since these keys can vary in location on different international keyboards. Allowed punctuation includes:

```
‘ ~ ! @ # $ % ^ & * ( ) - _ + = [ ] { } \ | ; : ’ " / ? . > , <
```

(4) These special keys can also be used:

Escape, Space, BackSpace, Tab, Linefeed, Clear, Return, Pause, Scroll_Lock, Sys_Req, Delete, Home, Left, Up, Right, Down, Prior, Page_Up, Next, Page_Down, End, Begin, Select, Print, Execute, Insert, Undo, Redo, Menu, Find, Cancel, Help, Break, Mode_switch, script_switch, Num_Lock,

F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, L1, F12, L2, F13, L3, F14, L4, F15, L5, F16, L6, F17, L7, F18, L8, F19, L9, F20, L10, F21, R1, F22, R2, F23, R3, F24, R4, F25, R5, F26, R6, F27, R7, F28, R8, F29, R9, F30, R10, F31, R11, F32, R12, F33, R13, F34, R14, F35, R15,

(5) For equivalents that work with the mouse buttons, use these:

Pointer_Left, Pointer_Right, Pointer_Up, Pointer_Down, Pointer_UpLeft, Pointer_UpRight, Pointer_DownLeft, Pointer_DownRight, Pointer_Button_Dflt, Pointer_Button1, Pointer_Button2, Pointer_Button3, Pointer_Button4, Pointer_Button5, Pointer_DblClick_Dflt, Pointer_DblClick1, Pointer_DblClick2, Pointer_DblClick3, Pointer_DblClick4, Pointer_DblClick5, Pointer_Drag_Dflt, Pointer_Drag1, Pointer_Drag2, Pointer_Drag3, Pointer_Drag4, Pointer_EnableKeys, Pointer_Accelerate, Pointer_DfltBtnNext, Pointer_DfltBtnPrev,

(6) Keypad keys are specified like this:

KP_Left, KP_Right, KP_Up, KP_Down, KP_Home, KP_Page_Up, KP_Page_Down, KP_End, KP_Insert, KP_Delete, KP_0, KP_1, KP_2, KP_3, KP_4, KP_5, KP_6, KP_7, KP_8, KP_9,

(7) These will also work but ignore the Shift modifier since they tend to appear in different locations on international keyboards:

KP_Space, KP_Tab, KP_Enter, KP_F1, KP_F2, KP_F3, KP_F4, KP_Prior, KP_Next, KP_Begin, KP_Insert, KP_Delete, KP_Equal, KP_Multiply, KP_Add, KP_Separator, KP_Subtract, KP_Decimal, KP_Divide,

exclam, quotedbl, numbersign, dollar, percent, ampersand, apostrophe, quoteright, parenleft, parenright, asterisk, plus, comma, minus, period, slash, colon, semicolon, less, equal, greater, question, at, bracketleft, backslash, bracketright, asciicircum, underscore, grave, quoteleft, braceleft, bar, braceright,

EuroSign, EcuSign, ColonSign, CruzeiroSign, FFrancSign, LiraSign, MillSign, NairaSign, PesetaSign, RupeeSign, WonSign, NewSheqelSign, DongSign,

Many other key names are available for use on international or special purpose keyboards:

asciitilde, nobreakspace, exclamdown, cent, sterling, currency, yen, brokenbar, section, diaeresis, copyright, ordfeminine, guillemotleft, notsign, hyphen, registered, macron, degree, plusminus, twosuperior, threesuperior, acute, mu, paragraph, periodcentered,

cedilla, onesuperior, masculine, guillemotright, onequarter, onehalf, threequarters, questiondown,

leftradical, topleftradical, horizconnector, topintegral, botintegral, vertconnector, topleftsqbracket, botleftsqbracket, toprightsqbracket, botrightsqbracket, topleftparens, botleftparens, toprightparens, botrightparens, leftmiddlecurlybrace, rightmiddlecurlybrace, topleftsummation, botleftsummation, topvertsummationconnector, botvertsummationconnector, toprightsummation, botrightsummation, rightmiddlesummation, lessthanequal, notequal, greaterthanequal, integral, therefore, variation, infinity, nabla, approximate, similarequal, ifonlyif, implies, identical, radical, includedin, includes, intersection, union, logicaland, logicalor, partialderivative, function, leftarrow, uparrow, rightarrow, downarrow, blank, soliddiamond, checkerboard, ht, ff, cr, lf, nl, vt, lowrightcorner, uprightcorner, upleftcorner, lowleftcorner, crossinglines, horizlinescan1, horizlinescan3, horizlinescan5, horizlinescan7, horizlinescan9, leftt, rightt, bott, topt, vertbar, emspace, enspace, em3space, em4space, digitspace, punctspace, thinspace, hairspace, emdash, endash, signifblank, ellipsis, doubbaselinedot, onethird, twothirds, onefifth, twofifths, threefifths, fourfifths, onesixth, fivesixths, careof, figdash, leftanglebracket, decimalpoint, rightanglebracket, marker, oneeighth, threeeighths, fiveeighths, seveneighths, trademark, signaturemark, trademarkincircle, leftopentriangle, rightopentriangle, emopencircle, emopenrectangle, leftsinglequotemark, rightsinglequotemark, leftdoublequotemark, rightdoublequotemark, prescription, minutes, seconds, latincross, hexagram, filledrectbullet, filledlefttribullet, filledrighttribullet, emfilledcircle, emfilledrect, enopencircbullet, enopensquarebullet, openrectbullet, opentribulletup, opentribulletdown, openstar, enfilledcircbullet, enfilledsqbullet, filledtribulletup, filledtribulletdown, leftpointer, rightpointer, club, diamond, heart, maltesecross, dagger, doubledagger, checkmark, ballotcross, musicalsharp, musicalflat, malesymbol, femalesymbol, telephone, telephonerecorder, phonographcopyright, caret, singlelowquotemark, doublelowquotemark, cursor, leftcaret, rightcaret, downcaret, upcaret, overbar, downtack, upshoe, downstile, underbar, jot, quad, uptack, circle, upstile, downshoe, rightshoe, leftshoe, lefttack, righttack,

Multi_key, Codeinput, SingleCandidate, MultipleCandidate, PreviousCandidate, Kanji, Muhenkan, Henkan_Mode, Henkan, Romaji, Hiragana, Katakana, Hiragana_Katakana, Zenkaku, Hankaku, Zenkaku_Hankaku, Touroku, Massyo, Kana_Lock, Kana_Shift, Eisu_Shift, Eisu_toggle, Kanji_Bangou, Zen_Koho, Mae_Koho,

ISO_Lock, ISO_Level2_Latch, ISO_Level3_Shift, ISO_Level3_Latch, ISO_Level3_Lock, ISO_Group_Shift, ISO_Group_Latch, ISO_Group_Lock, ISO_Next_Group, ISO_Next_Group_Lock, ISO_Prev_Group, ISO_Prev_Group_Lock, ISO_First_Group, ISO_First_Group_Lock, ISO_Last_Group, ISO_Last_Group_Lock, ISO_Left_Tab, ISO_Move_Line_Up, ISO_Move_Line_Down, ISO_Partial_Line_Up, ISO_Partial_Line_Down, ISO_Partial_Space_Left, ISO_Partial_Space_Right, ISO_Set_Margin_Left, ISO_Set_Margin_Right, ISO_Release_Margin_Left, ISO_Release_Margin_Right, ISO_Release_Both_Margins, ISO_Fast_Cursor_Left, ISO_Fast_Cursor_Right,

ISO_Fast_Cursor_Up, ISO_Fast_Cursor_Down, ISO_Continuous_Underline, ISO_Discontinuous_Underline, ISO_Emphasize, ISO_Center_Object, ISO_Enter

dead_grave, dead_acute, dead_circumflex, dead_tilde, dead_macron, dead_breve, dead_abovedot, dead_diaeresis, dead_abovering, dead_doubleacute, dead_caron, dead_cedilla, dead_ogonek, dead_iota, dead_voiced_sound, dead_semivoiced_sound, dead_belowdot,

First_Virtual_Screen, Prev_Virtual_Screen, Next_Virtual_Screen, Last_Virtual_Screen, Terminate_Server, AccessX_Enable, AccessX_Feedback_Enable, RepeatKeys_Enable, SlowKeys_Enable, BounceKeys_Enable, StickyKeys_Enable, MouseKeys_Enable, MouseKeys_Accel_Enable, Overlay1_Enable, Overlay2_Enable, AudibleBell_Enable, Pointer_Left, Pointer_Right, Pointer_Up,

_3270_Duplicate, _3270_FieldMark, _3270_Right2, 3270_Left2, _3270_BackTab, __3270_EraseEOF, _3270_EraseInput, 3270_Reset, _3270_Quit, _3270_PA1, __3270_PA2, _3270_PA3, _3270_Test, 3270_Attn, _3270_CursorBlink, __3270_AltCursor, _3270_KeyClick, 3270_Jump, _3270_Ident, _3270_Rule, __3270_Copy, _3270_Play, _3270_Setup, 3270_Record, _3270_ChangeScreen, __3270_DeleteWord, _3270_ExSelect, 3270_CursorSelect, _3270_PrintScreen, __3270_Enter,

Agrave, Aacute, Acircumflex, Atilde, Adiaeresis, Aring, AE, Ccedilla, Egrave, Eacute, Ecircumflex, Ediaeresis, Igrave, Iacute, Icircumflex, Idiaeresis, ETH, Eth, Ntilde, Ograve, Oacute, Ocircumflex, Otilde, Odiaeresis, multiply, Ooblique, Ugrave, Uacute, Ucircumflex, Udiaeresis, Yacute, THORN, Thorn, ssharp, agrave, aacute, acircumflex, atilde, adiaeresis, aring, ae, ccedilla, egrave, eacute, ecircumflex, ediaeresis, igrave, iacute, icircumflex, idiaeresis, eth, ntilde, ograve, oacute, ocircumflex, otilde, odiaeresis, division, oslash, ugrave, uacute, ucircumflex, udiaeresis, yacute, thorn, ydiaeresis, Aogonek, breve, Lstroke, Lcaron, Sacute, Scaron, Scedilla, Tcaron, Zacute, Zcaron, Zabovedot, aogonek, ogonek, lstroke, lcaron, sacute, caron, scaron, scedilla, tcaron, zacute, doubleacute, zcaron, zabovedot, Racute, Abreve, Lacute, Cacute, Ccaron, Eogonek, Ecaron, Dcaron, Dstroke, Nacute, Ncaron, Odoubleacute, Rcaron, Uring, Udoubleacute, Tcedilla, racute, abreve, lacute, cacute, ccaron, eogonek, ecaron, dcaron, dstroke, nacute, ncaron, odoubleacute, udoubleacute, rcaron, uring, tcedilla, abovedot, Hstroke, Hcircumflex, Iabovedot, Gbreve, Jcircumflex, hstroke, hcircumflex, idotless, gbreve, jcircumflex, Cabovedot, Ccircumflex, Gabovedot, Gcircumflex, Ubreve, Scircumflex, cabovedot, ccircumflex, gabovedot, gcircumflex, ubreve, scircumflex, kra, kappa, Rcedilla, Itilde, Lcedilla, Emacron, Gcedilla, Tslash, rcedilla, itilde, lcedilla, emacron, gcedilla, tslash, ENG, eng, Amacron, Iogonek, Eabovedot, Imacron, Ncedilla, Omacron, Kcedilla, Uogonek, Utilde, Umacron, amacron, iogonek, eabovedot, imacron, ncedilla, omacron, kcedilla, uogonek, utilde, umacron, OE, oe, Ydiaeresis, overline,

kana_fullstop, kana_openingbracket, kana_closingbracket, kana_comma, kana_conjunctive, kana_middledot, kana_WO, kana_a, kana_i, kana_u, kana_e, kana_o, kana_ya, kana_yu, kana_yo, kana_tsu, kana_tu, prolongedsound, kana_A, kana_I, kana_U,

kana_E, kana_O, kana_KA, kana_KI, kana_KU, kana_KE, kana_KO, kana_SA, kana_SHI, kana_SU, kana_SE, kana_SO, kana_TA, kana_CHI, kana_TI, kana_TSU, kana_TU, kana_TE, kana_TO, kana_NA, kana_NI, kana_NU, kana_NE, kana_NO, kana_HA, kana_HI, kana_FU, kana_HU, kana_HE, kana_HO, kana_MA, kana_MI, kana_MU, kana_ME, kana_MO, kana_YA, kana_YU, kana_YO, kana_RA, kana_RI, kana_RU, kana_RE, kana_RO, kana_WA, kana_N, voicedsound, semivoicedsound, kana_switch,

Arabic_comma, Arabic_semicolon, Arabic_question_mark, Arabic_hamza, Arabic_maddaonalef, Arabic_hamzaonalef, Arabic_hamzaonwaw, Arabic_hamzaunderalef, Arabic_hamzaonyeh, Arabic_alef, Arabic_beh, Arabic_tehmarbuta, Arabic_teh, Arabic_theh, Arabic_jeem, Arabic_hah, Arabic_khah, Arabic_dal, Arabic_thal, Arabic_ra, Arabic_zain, Arabic_seen, Arabic_sheen, Arabic_sad, Arabic_dad, Arabic_tah, Arabic_zah, Arabic_ain, Arabic_ghain, Arabic_tatweel, Arabic_feh, Arabic_qaf, Arabic_kaf, Arabic_lam, Arabic_meem, Arabic_noon, Arabic_ha, Arabic_heh, Arabic_waw, Arabic_alefmaksura, Arabic_yeh, Arabic_fathatan, Arabic_dammatan, Arabic_kasratan, Arabic_fatha, Arabic_damma, Arabic_kasra, Arabic_shadda, Arabic_sukun, Arabic_switch,

Serbian_dje, Macedonia_gje, Cyrillic_io, Ukrainian_ie, Ukranian_je, Macedonia_dse, Ukrainian_i, Ukranian_i, Ukrainian_yi, Ukranian_yi, Cyrillic_je, Serbian_je, Cyrillic_lje, Serbian_lje, Cyrillic_nje, Serbian_nje, Serbian_tshe, Macedonia_kje, Ukrainian_ghe_with_upturn, Byelorussian_shortu, Cyrillic_dzhe, Serbian_dze, numerosign, Serbian_DJE, Macedonia_GJE, Cyrillic_IO, Ukrainian_IE, Ukranian_JE, Macedonia_DSE, Ukrainian_I, Ukranian_I, Ukrainian_YI, Ukranian_YI, Cyrillic_JE, Serbian_JE, Cyrillic_LJE, Serbian_LJE, Cyrillic_NJE, Serbian_NJE, Serbian_TSHE, Macedonia_KJE, Ukrainian_GHE_WITH_UPTURN, Byelorussian_SHORTU, Cyrillic_DZHE, Serbian_DZE, Cyrillic_yu, Cyrillic_a, Cyrillic_be, Cyrillic_tse, Cyrillic_de, Cyrillic_ie, Cyrillic_ef, Cyrillic_ghe, Cyrillic_ha, Cyrillic_i, Cyrillic_shorti, Cyrillic_ka, Cyrillic_el, Cyrillic_em, Cyrillic_en, Cyrillic_o, Cyrillic_pe, Cyrillic_ya, Cyrillic_er, Cyrillic_es, Cyrillic_te, Cyrillic_u, Cyrillic_zhe, Cyrillic_ve, Cyrillic_softsign, Cyrillic_yeru, Cyrillic_ze, Cyrillic_sha, Cyrillic_e, Cyrillic_shcha, Cyrillic_che, Cyrillic_hardsign, Cyrillic_YU, Cyrillic_A, Cyrillic_BE, Cyrillic_TSE, Cyrillic_DE, Cyrillic_IE, Cyrillic_EF, Cyrillic_GHE, Cyrillic_HA, Cyrillic_I, Cyrillic_SHORTI, Cyrillic_KA, Cyrillic_EL, Cyrillic_EM, Cyrillic_EN, Cyrillic_O, Cyrillic_PE, Cyrillic_YA, Cyrillic_ER, Cyrillic_ES, Cyrillic_TE, Cyrillic_U, Cyrillic_ZHE, Cyrillic_VE, Cyrillic_SOFTSIGN, Cyrillic_YERU, Cyrillic_ZE, Cyrillic_SHA, Cyrillic_E, Cyrillic_SHCHA, Cyrillic_CHE, Cyrillic_HARDSIGN,

Greek_ALPHAaccent, Greek_EPSILONaccent, Greek_ETAaccent, Greek_IOTAaccent, Greek_IOTAdiaeresis, Greek_OMICRONaccent, Greek_UPSILONaccent, Greek_UPSILONdieresis, Greek_OMEGAaccent, Greek_accentdieresis, Greek_horizbar, Greek_alphaaccent, Greek_epsilonaccent, Greek_etaaccent, Greek_iotaaccent, Greek_iotadieresis, Greek_iotaaccentdieresis, Greek_omicronaccent, Greek_upsilonaccent, Greek_upsilondieresis, Greek_upsilonaccentdieresis,

Greek_omegaaccent, Greek_ALPHA, Greek_BETA, Greek_GAMMA, Greek_DELTA, Greek_EPSILON, Greek_ZETA, Greek_ETA, Greek_THETA, Greek_IOTA, Greek_KAPPA, Greek_LAMDA, Greek_LAMBDA, Greek_MU, Greek_NU, Greek_XI, Greek_OMICRON, Greek_PI, Greek_RHO, Greek_SIGMA, Greek_TAU, Greek_UPSILON, Greek_PHI, Greek_CHI, Greek_PSI, Greek_OMEGA, Greek_alpha, Greek_beta, Greek_gamma, Greek_delta, Greek_epsilon, Greek_zeta, Greek_eta, Greek_theta, Greek_iota, Greek_kappa, Greek_lamda, Greek_lambda, Greek_mu, Greek_nu, Greek_xi, Greek_omicron, Greek_pi, Greek_rho, Greek_sigma, Greek_finalsmallsigma, Greek_tau, Greek_upsilon, Greek_phi, Greek_chi, Greek_psi, Greek_omega, Greek_switch,

hebrew_doublelowline, hebrew_aleph, hebrew_bet, hebrew_beth, hebrew_gimel, hebrew_gimmel, hebrew_dalet, hebrew_daleth, hebrew_he, hebrew_waw, hebrew_zain, hebrew_zayin, hebrew_chet, hebrew_het, hebrew_tet, hebrew_teth, hebrew_yod, hebrew_finalkaph, hebrew_kaph, hebrew_lamed, hebrew_finalmem, hebrew_mem, hebrew_finalnun, hebrew_nun, hebrew_samech, hebrew_samekh, hebrew_ayin, hebrew_finalpe, hebrew_pe, hebrew_finalzade, hebrew_finalzadi, hebrew_zade, hebrew_zadi, hebrew_qoph, hebrew_kuf, hebrew_resh, hebrew_shin, hebrew_taw, hebrew_taf, Hebrew_switch,

Thai_kokai, Thai_khokhai, Thai_khokhuat, Thai_khokhwai, Thai_khokhon, Thai_khorakhang, Thai_ngongu, Thai_chochan, Thai_choching, Thai_chochang, Thai_soso, Thai_chochoe, Thai_yoying, Thai_dochada, Thai_topatak, Thai_thothan, Thai_thonangmontho, Thai_thophuthao, Thai_nonen, Thai_dodek, Thai_totao, Thai_thothung, Thai_thothahan, Thai_thothong, Thai_nonu, Thai_bobaimai, Thai_popla, Thai_phophung, Thai_fofa, Thai_phophan, Thai_fofan, Thai_phosamphao, Thai_moma, Thai_yoyak, Thai_rorua, Thai_ru, Thai_loling, Thai_lu, Thai_wowaen, Thai_sosala, Thai_sorusi, Thai_sosua, Thai_hohip, Thai_lochula, Thai_oang, Thai_honokhuk, Thai_paiyannoi, Thai_saraa, Thai_maihanakat, Thai_saraaa, Thai_saraam, Thai_sarai, Thai_saraii, Thai_saraue, Thai_sarauee, Thai_sarau, Thai_sarauu, Thai_phinthu, Thai_maihanakat_maitho, Thai_baht, Thai_sarae, Thai_saraae, Thai_sarao, Thai_saraaimaimuan, Thai_saraaimaimalai, Thai_lakkhangyao, Thai_maiyamok, Thai_maitaikhu, Thai_maiek, Thai_maitho, Thai_maitri, Thai_maichattawa, Thai_thanthakhat, Thai_nikhahit, Thai_leksun, Thai_leknung, Thai_leksong, Thai_leksam, Thai_leksi, Thai_lekha, Thai_lekhok, Thai_lekchet, Thai_lekpaet, Thai_lekkao,

Hangul, Hangul_Start, Hangul_End, Hangul_Hanja, Hangul_Jamo, Hangul_Romaja, Hangul_Codeinput, Hangul_Jeonja, Hangul_Banja, Hangul_PreHanja, Hangul_PostHanja, Hangul_SingleCandidate, Hangul_MultipleCandidate, Hangul_PreviousCandidate, Hangul_Special, Hangul_switch, Hangul_Kiyeog, Hangul_SsangKiyeog, Hangul_KiyeogSios, Hangul_Nieun, Hangul_NieunJieuj, Hangul_NieunHieuh, Hangul_Dikeud, Hangul_SsangDikeud, Hangul_Rieul, Hangul_RieulKiyeog, Hangul_RieulMieum, Hangul_RieulPieub, Hangul_RieulSios,

Hangul_RieulTieut, Hangul_RieulPhieuf, Hangul_RieulHieuh, Hangul_Mieum, Hangul_Pieub, Hangul_SsangPieub, Hangul_PieubSios, Hangul_Sios, Hangul_SsangSios, Hangul_Ieung, Hangul_Jieuj, Hangul_SsangJieuj, Hangul_Cieuc, Hangul_Khieuq, Hangul_Tieut, Hangul_Phieuf, Hangul_Hieuh, Hangul_A, Hangul_AE, Hangul_YA, Hangul_YAE, Hangul_EO, Hangul_E, Hangul_YEO, Hangul_YE, Hangul_O, Hangul_WA, Hangul_WAE, Hangul_OE, Hangul_YO, Hangul_U, Hangul_WEO, Hangul_WE, Hangul_WI, Hangul_YU, Hangul_EU, Hangul_YI, Hangul_I, Hangul_J_Kiyeog, Hangul_J_SsangKiyeog, Hangul_J_KiyeogSios, Hangul_J_Nieun, Hangul_J_NieunJieuj, Hangul_J_NieunHieuh, Hangul_J_Dikeud, Hangul_J_Rieul, Hangul_J_RieulKiyeog, Hangul_J_RieulMieum, Hangul_J_RieulPieub, Hangul_J_RieulSios, Hangul_J_RieulTieut, Hangul_J_RieulPhieuf, Hangul_J_RieulHieuh, Hangul_J_Mieum, Hangul_J_Pieub, Hangul_J_PieubSios, Hangul_J_Sios, Hangul_J_SsangSios, Hangul_J_Ieung, Hangul_J_Jieuj, Hangul_J_Cieuc, Hangul_J_Khieuq, Hangul_J_Tieut, Hangul_J_Phieuf, Hangul_J_Hieuh, Hangul_RieulYeorinHieuh, Hangul_SunkyeongeumMieum, Hangul_SunkyeongeumPieub, Hangul_PanSios, Hangul_KkogjiDalrinIeung, Hangul_SunkyeongeumPhieuf, Hangul_YeorinHieuh, Hangul_AraeA, Hangul_AraeAE, Hangul_J_PanSios, Hangul_J_KkogjiDalrinIeung, Hangul_J_YeorinHieuh, Korean_Won,

Armenian_eternity, Armenian_section_sign, Armenian_full_stop, Armenian_verjaket, Armenian_parenright, Armenian_parenleft, Armenian_guillemotright, Armenian_guillemotleft, Armenian_em_dash, Armenian_dot, Armenian_mijaket, Armenian_separation_mark, Armenian_but, Armenian_comma, Armenian_en_dash, Armenian_hyphen, Armenian_yentamna, Armenian_ellipsis, Armenian_exclam, Armenian_amanak, Armenian_accent, Armenian_shesht, Armenian_question, Armenian_paruyk, Armenian_AYB, Armenian_ayb, Armenian_BEN, Armenian_ben, Armenian_GIM, Armenian_gim, Armenian_DA, Armenian_da, Armenian_YECH, Armenian_yech, Armenian_ZA, Armenian_za, Armenian_E, Armenian_e, Armenian_AT, Armenian_at, Armenian_TO, Armenian_to, Armenian_ZHE, Armenian_zhe, Armenian_INI, Armenian_ini, Armenian_LYUN, Armenian_lyun, Armenian_KHE, Armenian_khe, Armenian_TSA, Armenian_tsa, Armenian_KEN, Armenian_ken, Armenian_HO, Armenian_ho, Armenian_DZA, Armenian_dza, Armenian_GHAT, Armenian_ghat, Armenian_TCHE, Armenian_tche, Armenian_MEN, Armenian_men, Armenian_HI, Armenian_hi, Armenian_NU, Armenian_nu, Armenian_SHA, Armenian_sha, Armenian_VO, Armenian_vo, Armenian_CHA, Armenian_cha, Armenian_PE, Armenian_pe, Armenian_JE, Armenian_je, Armenian_RA, Armenian_ra, Armenian_SE, Armenian_se, Armenian_VEV, Armenian_vev, Armenian_TYUN, Armenian_tyun, Armenian_RE, Armenian_re, Armenian_TSO, Armenian_tso, Armenian_VYUN, Armenian_vyun, Armenian_PYUR, Armenian_pyur, Armenian_KE, Armenian_ke, Armenian_O, Armenian_o, Armenian_FE, Armenian_fe, Armenian_apostrophe, Armenian_ligature_ew,

Georgian_an, Georgian_ban, Georgian_gan, Georgian_don, Georgian_en, Georgian_vin, Georgian_zen, Georgian_tan, Georgian_in, Georgian_kan, Georgian_las, Georgian_man,

Georgian_nar, Georgian_on, Georgian_par, Georgian_zhar, Georgian_rae, Georgian_san, Georgian_tar, Georgian_un, Georgian_phar, Georgian_khar, Georgian_ghan, Georgian_qar, Georgian_shin, Georgian_chin, Georgian_can, Georgian_jil, Georgian_cil, Georgian_char, Georgian_xan, Georgian_jhan, Georgian_hae, Georgian_he, Georgian_hie, Georgian_we, Georgian_har, Georgian_hoe, Georgian_fi,

2.5. File Sets

Wing provides a way to define sets of files that can be used in various ways within the IDE, such as for searching particular batches of files and adding only certain kinds of files to a project.

To view or alter the defined file sets, use the `File Sets...` item in the File menu. This will display a file set editor within the Preferences manager.

When adding or editing a file set, the following information may be entered:

- **Name** -- The name of the file set
- **Includes** -- A list of inclusion criteria, each of which contains a type and a specification. A file will be included in the file set if any one of these include criteria matches it.
- **Excludes** -- A list of exclusion criteria, any of which can match to cause a file to be excluded from the file set even if one or more include matches were also found.

The following types of include and exclude criteria are supported:

- **Wildcard on Filename** -- The specification in this case is a wildcard that must match the file name. The wildcards supported are those provided by Python's fnmatch module.
- **Wildcard on Directory Name** -- The specification in this case is a wildcard that must match the directory name.
- **Mime Type** -- The specification in this case names a MIME type supported by Wing IDE. If additional file extensions need to be mapped to a MIME type, use the **Extra File Types** preference to define them.

Once defined, file sets are presented by name in the **Search in Files** tool's batch search facility and in the **Project** tool's batch file addition features.

Any problems encountered in using the file sets are reported in the Messages area.

Project Manager

The project manager provides a convenient index of the files in your software project and collects information needed by Wing's debugger, source code analysis tools, and other facilities.

To get the most out of Wing's debugger and source analysis engine, you must add your source base to your project file and may in some cases need to set up `PYTHONPATH` and other values in **Project-Wide Properties** and/or **Per-File Properties**.

3.1. Creating a Project

To create a new project, use the New Project item in the Project menu. This will prompt you to save any changes to your currently open project and will create a new blank project. If Wing is started without any command line arguments, the most recent project is opened, or if no project exists then the Default Project is opened.

To add files to your project, use the following items in the Project menu:

- **Add Current File** will add the frontmost current open file to the project if it is not already there.
- **Add New File** is used to create a new file and simultaneously add it to your project.
- **Add File** will prompt you to select a single file to add to the project view. Note that this also may result in adding a new directory to the project manager window, if that file is the first to be added for a directory.
- **Add Package** can be used to add more than one file at once. Select a directory with your left mouse button so that the directory name is shown in the area at the bottom of the file selection window. Then click OK. You will be prompted with a list of files within the selected directory. Highlight any that you wish to add, using shift-click to highlight a contiguous range or ctrl-click to select an arbitrary set. Then click Yes to add all those files to your project.

- **Add Directory Tree** can be used to add many files in a directory structure in one operation. Select a directory from the provided list. Once this dialog is accepted, Wing will add files recursively from the selected directory and all its children.

A subset of these options be accessed from the popup menu that appears when right-clicking your mouse on the surface of the project manager window.

3.2. Removing Files and Packages

To remove a specific file, select it and use the Remove From Project menu item in the right-click popup menu from the surface of the Project Manager window, or by selecting an item on the project and using Remove Selected Entry in the Project menu. You can also remove a whole directory and all the files that it contains in this way.

3.3. Saving the Project

Once a project file has been saved the first time, it will be auto-saved whenever you close the project, start a debug session, or exit Wing. This behavior can be turned off with the **Save without Asking** preference.

You can also save a copy of your project to another location or name with `Save Project As...` in the Project menu.

> **Moving Project Files**
> Using `Save Project As...` is recommended if you need to alter the location of your project file in relation to your source files because it will update the partial relative paths that the project manager uses to locate files in the project. Otherwise, Wing may not be able to find all of the files in the project.

3.4. Sorting the View

The project can be set to show your files in one of several modes, using the `Options` menu in the top right of the project view:

- **By Flattened Tree** -- This view (the default) shows files organized according to their location on disk. Each directory is shown at the top level with path names

shown as partial relative paths based on the location of the project file. If you alter the location of the project file with `Save Project As...`, these paths will be updated accordingly.

- **By Tree** -- This displays the project files in true tree form. The tree structure is based on the partial relative path from the project file.
- **By Mime Type** -- This view organizes your files by MIME type.

3.5. Keyboard Navigation

Once it has the focus, the project manager tree view is navigable with the keyboard, using the up/down arrow keys, page up and page down, and home/end.

Use the right arrow key on a parent to display its children, or the left arrow key to hide them.

Holding down the shift key while pressing the right arrow will expand recursively below the expansion point. Recursive expansion is capped at five additional levels for each operation to avoid infinite recursion resulting from symbolic links.

Whenever a tree row is selected, pressing enter or return will open that item within Wing IDE.

3.6. Sharing Projects

Project File Types
There are two related formats in which you can save your project. One supports sharing the project file with other developers, via a revision control system or other method.

The default project type is 'Normal', which stores all project data into a single file. The file name for these files should end in `'.wpr'`.

To share a project file with other developers, change `Project Type` in the `Options` tab of `Project Properties` to `Shared (Two Files)`. Then save your project to obtain the two separate project files on disk. The main project file name ends in `'.wpr'` and will contain only shareable data. All user-specific data will be stored in a separate file with name ending in `'.wpu'`.

If you subsequently change from a Shared project back to Normal, the user-specific data will be merged back into the machine project file and the file ending in `'.wpu'` will be removed from disk.

Note that both the combined 'Normal' file and two split 'Shared' files use the same textual file format that is used for the preferences file. See section **Preferences File Format** for more information on the format itself.

3.7. Project-wide Properties

Each project has a set of top-level properties that can be accessed and edited via the **Properties** item in the Project menu. These can be used to configure the Python environment used by the **debugger** and the **source code analysis** engine, which drives Wing's auto completion, source index, and other capabilities. Project properties are also provided to set options for the project and to enable and configure extensions for revision control, Zope, and other tools.

Any string value for a property may contain environment variable references using the `$(name)` notation. Anything inside the parentheses will be interpreted as the name of an environment variable and will be replaced with the value of the environment variable when it used by the IDE. If the environment variable is not set, the reference will be replaced by an empty string. The system environment, as modified by the project-wide environment property (see below), is used to expand variable references.

Environment

To get the most out of Wing, it is important to set these values in the Environment tab correctly for your project:

Python Executable -- When the *Custom* radio button is checked and the entered field is non-blank, this can be used to set the full path to the Python executable that should be used when debugging source code in this project. When *Use default* is selected, Wing tries to use the default Python obtained by typing `python` on the command line. If this fails, Wing will search for Python in `/usr/local` and `/usr` (on Linux/Unix) or in the registry (on Windows).

Python Path -- The `PYTHONPATH` is used by Python to locate modules that are imported at runtime with the `import` statement. When the *Use default* checkbox in this area is checked, the inherited `PYTHONPATH` environment variable is used for debug sessions. Otherwise, when *Custom setting* is selected, the specified `PYTHONPATH` is used.

Environment -- This is used to specify values that should be added, modified, or

removed from the environment that is inherited by debug processes started from Wing IDE and is used to expand enviroment variable references used in other properties. Each entry is in `var=value` form and must be specified one per line in the provided entry area. An entry in the form `var=` (without a value) will remove the given variable to be undefined. Note that you are operating on the environment inherited by the IDE when it started and not modifying an empty environment. When the *Use system environment* choice is set, any entered values are ignored and the inherited environment is used without changes.

Debug

The following properties are defined in the Debug tab:

Initial Directory -- When the *Use default* radio button is checked, the initial working directory set for each debug session will be the location where the main entry point file or project file is located. Otherwise, when *Custom setting* is selected, the specified directory is used or, when blank, the project file's directory is used.

Build Command -- This command will be executed before starting debug on any source in this project. This is useful to make sure that C/C++ extension modules are built, for example in conjunction with an external `Makefile` or `distutils` script, before execution is started.

Options

These project options are provided:

Project Type -- This can be used to select whether or not the project will be shared among several developers. When shared, the project will be written to two files, one of which can be shared with other developers. See **Project Types** for details.

Preferred Line Ending and **Line Ending Policy** control whether or not the project prefers a particular line ending style (line feed, carriage return, or carriage return + line feed), and how to enforce that style, if at all. By default, projects do not enforce a line ending style but rather insert new lines to match any existing line endings in the file.

Preferred Indent Style and **Indent Style Policy** control whether or not the project prefers a particular type of indentation style for files (spaces only, tabs only, or mixed tabs and spaces), and how to enforce that style, if at all. By default, projects do not enforce an indent style but rather insert new lines to match any existing indentation in the file.

Extensions

The Extensions tab of Project Properties is used to control revision control and other add-ons on a per-project basis:

Enable Revision Control and **Revision Control System** are used to turn on a particular revision control integration for this project. Currently, only a minimalist CVS integration is available.

Enable Zope2/Plone Support and **Zope2 Instance Home** are used for Zope 2.x and Plone projects to provide the Instance Home directory used by Zope. This is needed because Zope 2.x implements import magic that works differently from Python's default `import` and thus adding the instance home directory to `PYTHONPATH` is not sufficient. Wing's source analyzer needs this extra clue to properly find and process the Zope instance-specific sources.

If you are sharing a project file with other developers through a revision control system, and have set Project Type `Shared`, it is important to note that the above values are stored in the private branch of the project file. This means that each developer needs to set these values independently, to match the specific environment on their development machine.

3.8. Per-file Properties

Per-file properties can be set by right-clicking on a source file and selecting the `Properties` menu item in the popup, by right-clicking on a file in the project view and selecting `File Properties`, or by opening a file and using the `Current File Properties...` item in the Source menu. For Debug and Python Settings, values entered here will override any corresponding project-wide values.

File Attributes

File Type -- This property specifies the file type for a given file, overriding the type determined automatically from its file extension and/or content. This property is recommended only when the **Extra File Types** preference cannot be used to specify encoding based on filename extension.

Encoding -- This can be used to specify the encoding with which a file will be saved. When it is altered, the file will be opened into an editor and the encoding will not change until it is saved to disk. If it is not saved, the encoding attribute will revert to its previous setting. The encoding cannot be altered with this property if it is being

defined by an encoding comment in a Python, HTML, XML, or gettext PO file. In this case, the file should be opened and the encoding comment changed. Wing will save the file under the newly specified encoding.

Important: Files saved under a different encoding without an encoding comment may not be readable by other editors because there is no way for them to determine the file's encoding if it differs from the system or disk default. Wing stores the selected encoding in the project file, but no mark is written in the file except for those encodings that naturally use a Byte Order Mark (BOM), such as utf_8, utf_16_le, utf_16_be, utf_32_le. utf_32_be.

Line Ending Style -- Specifies which type of line ending (line feed, carriage return, or carriage return and line feed) is used in the file. When altered, the file will be opened and changed in an editor. The change does not take effect until the file is saved to disk.

Indent Style -- This property can be used in non-Python files to change the type of indent entered into the file for newly added lines. For Python files, the only way to alter indentation in a file is with the **Indentation manager**.

Read-only on Disk -- This property reflects whether or not the file is marked read-only on disk. Altering it will change the file's disk protections for the owner of the file (on Posix, group/world permissions are never altered).

Editor

These properties define how the file is displayed in the editor:

Show Whitespace -- This allows overriding the **Show White Space** preference on a per-file basis.

Show EOL -- This allows overriding the **Show EOL** preference on a per-file basis.

Show Indent Guides -- This allows overriding the **Show Indent Guides** preference on a per-file basis.

Ignore Indent Errors -- Wing normally reports potentially serious indentation inconsistency in Python files. This property can be used to disable this check on a per-file basis (it is also available in the warning dialog).

Ignore EOL Errors -- When the project's **Line Ending Policy** is set to warn about line ending mismatches, this property can be used to disable warnings for a particular file.

Environment

These properties are the same as for the Python Settings defined in **Project-Wide Properties**. Values defined per-file override the corresponding project-wide property.

For the `Environment` attribute, note that the option menu area contains some additional choices. Use *add to project values* to apply the values specified here to the runtime environment specified by the project, or *add to system environment* to bypass the project-wide values and apply the per-file values directly to the environment set by the operating system.

Debug

The per-file debug properties dialog contains all the same fields described in **Project-Wide Properties**, with the following additions:

Run Arguments -- Enter any run arguments here. Wing does not interpret backslashes (") on the command line and passes them unchanged to the debug process. The only exceptions to this rule are `\'` and `\"` (backslash followed by single or double quote), which allow inclusion of quotes inside quoted multi-word arguments.

Show this dialog before each run -- Check this checkbox if you want the debug options dialog to appear each time you start a debug session.

Values defined per-file override or modify the corresponding project-wide property.

When debugging, only per-file debug properties set on the *initially invoked file* are used. Even if other files with set properties are used in the debug session, any values set for them will be ignored.

3.9. Navigating to Files

Files can be opened from the project manager window by double clicking or middle clicking on the file name, or right-clicking and using the Open in Wing IDE menu item.

Files may also be opened using an external viewer or editor by right-clicking on the file and using the Open in External Viewer item. On Windows and Mac OS X, this opens the file as if you had double clicked on it. On Linux, the preferences **File Display Commands** and **Extra Mime Types** can be used to configure how files are opened.

You can also execute Makefiles, python source, and any executable files by selecting the

Execute Selected File item from the popup menu. This executes outside of the debugger with any input/output occurring in the window from which Wing was launched (if any).

Source Code Editor

Wing IDE's source code editor is designed to make it easier to adopt the IDE even if you are used to other editors.

Editor Overview
Key things to know about the editor are:

- The editor has personalities that emulate other commonly used editors such as Visual Studio, VI/Vim, Emacs, and Brief.
- Context-appropriate auto-completion, goto-definition, and code index menus are available when working in Python code
- The editor supports a wide variety of file types for syntax colorization.
- Key mappings and many other behaviors are configurable.
- The editor supports structural folding for some file types

4.1. Syntax Colorization

The editor will attempt to colorize documents according to their MIME type, which is determined by the file extension, or content. For example, any file ending in `.py` will be colorized as a Python source code document. Any file whose MIME type cannot be determined will display all text in black normal font by default.

All the available colorization document types are listed in the File Properties dialog's Editor tab. If you have a file that is not being recognized automatically, you can use the `File Type` menu found there to alter the way the file is being displayed. Your selections from this menu are stored in your project file, so changes made are permanent in the context of that project.

If you have many files with an unrecognized extension, use the **Extra File Types** preference to add your extension.

4.2. Right-click Editor Menu

Editor Popup Menu
Right-clicking on the surface of the editor will display a context menu with commonly used commands such as Copy, Paste, Goto Definition, and indentation operations. User-defined scripts may also add items here, as described in the Scripting chapter.

4.3. Navigating Source

The set of menus at the top of the editor can be used to navigate through your source code. When editor tabs are hidden, the left-most of these menus lists the currently open files by name. The additional popup menus indicate the scope of the current cursor selection in the file and may be used to navigate within the top-level scope, or within sub-scopes when they exist.

You can also use the Goto Definition menu item in the editor context menu to click on a construct in your source and zoom to its point of definition. Alternatively, place the cursor or selection on a symbol and use the Goto Selected Symbol Defn item in the Source menu, or its keyboard equivalent.

4.4. File status and read-only files

The editor tabs, or editor selection menu when the tabs are hidden, indicate the status of the file by appending `*` when the file has been edited or `(r/o)` when the file is read-only. This information is mirrored for the current file in the status area at the bottom left of each editor window.

Files that are read-only on disk are initially opened within a read-only editor. Use the file's context menu (right-click) to toggle between read-only and writable state. This alters both the editability of the editor and the writability of the disk file.

4.5. Transient vs. non-Transient Editors

Wing can open files in two modes:

Transient Mode -- Files opened when searching, debugging, navigating to point of definition, and using the Project and Source Browser tools with the `Follow Selection` checkbox enabled are all opened in transient mode and will be automatically closed when hidden. The maximum number of non-visible transient files to keep open at any given time can be set with the **Editor / Advanced / Transient Threshold** preference.

Non-Transient Mode -- Files opened normally from the File menu, from the keyboard file selector, or by double clicking on items in the Project tool will be opened in non-transient mode, and kept open until they are explicitly closed. Transient files that are edited are also automatically converted to non-transient mode.

A file can be switched between transient and non-transient mode by clicking on the stick pin icon in the upper right of the editor area. Right-click on the stick pin icon to navigate to recently visited files (blue items were transient, black items non-transient).

4.6. Structural Folding

The editor supports optional structural folding for Python, C, C++, Java, Javascript, HTML, Eiffel, Lisp, Ruby, and a number of other file formats. This allows you to visually collapse logical hierarchical sections of your code while you are working in other parts of the file.

You can turn Structural Folding on and off as a whole with the **Enable Folding** preference.

The **Fold Line Mode** preference can be used to determine whether or not a horizontal line is drawn at fold points, whether it is drawn above or below the fold point, and whether it is shown when the fold point is collapsed or expanded. **Fold Indicator Style** is used to select the look of the fold marks shown at fold points.

Once folding is turned on, an additional margin appears to the left of source files, where fold points are displayed. Left mouse click on one of these marks to collapse or expand that fold point. Right mouse clicking anywhere on the fold margin displays a context menu with the various folding operations.

You can also hold down the following key modifiers while left-clicking to modify the folding behavior:

- **Shift** -- Clicking on any fold point while holding down the shift key will expand

that point and all its children recursively so that the maximum level of expansion is increased by one.

- **Ctrl** -- Clicking on any fold point while holding down the ctrl key will collapse that point and all its children recursively so that the maximum level of expansion is decreased by one.

- **Ctrl+Shift** -- On a currently expanded fold point, this will collapse all child fold points recursively to maximum depth, as well as just the outer one. When the fold point is subsequently re-expanded with a regular click, its children will appear collapsed. Ctrl-shift-click on a collapsed fold point will force re-expansion of all children recursively to maximum depth.

Fold commands are also available in the `Structural Folding` section of the `Source` menu, indicates the key equivalents assigned to the operations:

- **Toggle Current Fold** -- Like clicking on the fold margin, this operates on the first fold point found in the current selection or on current line.

- **Collapse Current More** -- Like ctrl-clicking, this collapses the current fold point one more level than it is now.

- **Expand Current More** -- Like shift-clicking, this expands the current fold point one more level than it is now.

- **Collapse Current Completely** -- Like shift-ctrl-clicking on an expanded node, this collapses all children recursively to maximum depth.

- **Expand Current Completely** -- Like shift-ctrl-clicking on a collapsed node, this ensures that all children are expanded recursively to maximum depth.

- **Collapse All** -- Unconditionally collapse the entire file recursively.

- **Expand All** -- Unconditionally expand the entire file recursively.

- **Fold Python Methods** -- Fold up all methods in all classes in the file.

- **Fold Python Classes** -- Fold up all classes in the file.

- **Fold Python Classes and Defs** -- Fold up all classes and any top-level function definitions in the file.

4.7. Brace Matching

Wing will highlight matching braces in green when the cursor is adjacent to a brace. Mismatched braces are highlighted in red.

You can cause Wing to select the entire contents of the innermost brace pair from the current cursor position with the Match Braces item in the Source menu.

Parenthesis, square brackets, and curly braces are matched in all files. Angle brackets (< and >) are matched also in HTML and XML files.

4.8. Indentation

Since indentation is syntactically significant in Python, Wing provides a range of features for inspecting and managing indentation in source code.

Indentation Preferences

The following preferences affect how the indentation features behave:

1) The **Use Indent Analysis** preference is used to control whether analysis of current file content is used to determine the type of indentation placed during edits. It can be enabled for all files, only for Python files, or disabled.

2) The **Default Tab Size** preference defines the position of tab stops and is used to determine the rendering of files with tabs only, or non-Python files with mixed tab and space indentation. In Python files with mixed indents, this value is ignored and the file is always shown in the way that the Python interpreter would see it.

3) The **Default Indent Size** preference defines the default size of each level of indent, in spaces. This is used in new empty files or when indent analysis has been disabled. Wing may override this value in files that contain only tabs in indentation, in order to make it a multiple of the configured tab size.

4) The **Default Indent Style** preference defines the default indentation style, one of `spaces-only`, `tabs-only`, or `mixed`. This is used in new empty files or when indent analysis has been disabled. Mixed indentation replaces each tab-size spaces with one tab character.

These preferences define how indentation is handled by the editor:

5) The **Auto-Indent** preference controls whether or not each new line is automatically indented.

6) The **Show Indent Guides** preference controls whether or not to show indentation guides as light vertical lines. This value can be overridden on a file-by-file basis from Editor tab in File Properties.

7) The **Show Python Indent Warnings** preference can be used to enable or disable warnings for Python files that may contain confusing or damaged indentation.

8) The **Show Override Warnings** preference controls whether or not Wing shows a warnings when the user enters indentation that does not match the form already within a file. This is currently only possible in non-Python files, by altering the `Indent Style` attribute in `File Properties`.

How Indent Style is Determined

When an existing file is opened, it is scanned to determine what type of indentation is used in that file. If the file contains some indentation, this may override the tab size, indent size, and indent style values given in preferences and the file will be indented in a way that matches its existing content rather than with your configured defaults. If mixed forms of indentation are found, the most common form is used.

For Python files, the detected form of indentation cannot be overridden and Wing will always match new indentation to existing indentation. Mixing indentation styles in Python is dangerous because indentation has syntactic importance.

For non-Python files, you can change indentation style on the fly using the `Indent Style` property in the `File Properties` dialog. This allows creating files that intentionally mix indentation forms in different parts of the file. To ask Wing to return to the form of indentation it determines as most prominent in the file, select `Match Existing Indents`.

You may also convert the entire file between different forms of indentation using the Indentation Manager available from the Indentation portion of the Source menu. This is described in **Indentation Manager**.

Tab Size
Tab size is automatically forced to 8 characters for all Python source files that contain some spaces in indentation. This is done because the Python interpreter defines tabs as 8 characters in size when used together with spaces. This version of Wing does not recognize `vi` style tab size comments, but it does apply the **Tab Size** preference when a file contains only tabs in indentation, or if it is a non-Python file.

Indentation Policy

The project manager also provides the ability to define the preferred indentation style (overriding the preference-defined style) and to specify a policy for enforcing line endings, on a per-project basis. This is accomplished with **Preferred Line Ending** and **Line Ending Policy** under Options in Project Properties.

4.8.1. Auto-Indent

The IDE ships with auto-indent turned on. This causes leading white space to be added to each newly created line, as return or enter are pressed. Enough white space is inserted to match the indentation level of the previous line, possibly adding or removing a level of indentation if this is indicated by context in the source (such as `if`, `while`, or `return`).

Note that if preference **Auto-indent** is set to `false`, auto-indent does not occur until the tab key is pressed.

4.8.2. The Tab Key

By default, the tab key either indents according to context or increases the indent depth at the current cursor position by one level (this depends on the selected editor **Personality**). If one or more lines are selected, this instead operates on the indentation of all selected lines by one level.

To insert a real tab character regardless of the indentation mode or the position of the cursor on a line, type Ctrl-Tab or Ctrl-T.

The behavior of the tab key can be altered using the **Tab Key Action** preference.

4.8.3. Checking Indentation

Wing IDE analyses existing indentation whenever it opens a Python source file, and will indicate a potentially problematic mix of indentation styles, allowing you to attempt to repair the file. Files can be inspected more closely or repaired at any time using the **Indentation Manager**.

To turn off indentation warnings in Python files, use the **Show Python Indent Warnings** preference.

Wing also indicates suspiciously mismatched indentation in source code by underlining the indent area of the relevant lines in blue.

In general, mixing tab/space and space-only indentation in the same file can be confusing, especially if files are viewed with different editors and by different developers. We recommend using spaces only or tabs only as the best alternatives. To convert existing code containing a mix of tabs and spaces, use the Indentation Manager.

4.8.4. Changing Block Indentation

Wing provides `Indent` and `Outdent` commands in the `Indentation` portion of the Source menu, to support increasing or decreasing the level of indentation for selected blocks of text. All lines that are included in the current text selection are moved, even if the entire line isn't selected.

Indentation placed by these commands will contain either only spaces, only tabs, or a mixture of tabs and spaces, as determined by the method described in **Indentation**.

Indenting to Match
The command `Indent Lines to Match` (also in the `Indentation` sub-menu) will indent or outdent the current line or selected lines to the level as a unit so that the first line is positioned as it would have been positioned by Wing's auto-indentation facility. This is very useful when moving around blocks of code.

4.8.5. Indentation Manager

The Indentation manager, accessible from the `Tools` menu, can be used to inspect and change indentation style in source files. It has two parts: (1) The indentation report, and (2) the indentation converter.

A report on the nature of existing indentation found in your source file is given above the horizontal divider. This includes the number of spaces-only, tabs-only, and mixed tabs-and-space indents found, information about whether indentation in the file may be problematic to the Python interpreter, and the tab and indent size computed for that file. The manager also provides information about where the computed tab and indent size value come from (for example, an empty file results in use of the defaults configured in preferences).

Conversion options for your file are given below the horizontal divider. The three tabs are used to select the type of conversion desired, and each tab contains information about the availability and action of that conversion, and a button to start the conversion. Most of these conversions have no parameters that can be altered by the user. Only in the

case of conversion from tabs-only to spaces-only indentation styles, the tab size value shown in the indentation report is made editable, so that the configured default can be overridden.

Once conversion is complete, the indentation manager updates to display the new status of the file, and action of any subsequent conversions.

4.9. Auto-completion

While typing in Python source code, Wing will display a context-appropriate auto-completion list. To use it, type until the correct symbol is hilighted in the list and then press the tab key. Wing will fill in the remaining characters for the source symbol, correcting any spelling errors you might have made in the name.

If you wish to select a name without typing enough of the name to make your selection unique in the auto-completer, you can use the keyboard up and down arrow keys or mouse to scroll up and down in the popup list. Pressing tab or double clicking on a list item will complete that symbol in your source.

To cancel out of the auto-completion popup, press the `Esc` key or `ctrl-g`. The auto-completer will also disappear when you exit the source symbol (for example, by pushing `space` or any other character that can't be contained in a source symbol), if you click elsewhere on the surface of the source code, or if you issue other keyboard bound commands that are not accepted by the auto-completer (for example, `save` through keyboard equivalent or right/left arrow keys).

Auto-completion Limitations
Auto-completion covers most but not all possible scenarios at this time. See section **Source Code Analysis** for more information on current capabilities.

4.10. Source Assistant

Wing's `Source Assistant` tool can be used while viewing or editing source code to display additional information about the point of definition of source constructs located near the current insertion cursor position.

The following data is displayed in the source assistant:

- **File** -- The file and line number where the source symbol is defined

- **Call Signature** -- The function or method name, arguments, and return value if known for the source construct. The class name is also included if available. This value is blank for non-callable values.
- **Doc String** -- The documentation string for the source symbol, if available

Consider the following example where the `|` at the end of `myfile2` represents the insertion cursor:

myfile1.py:

```
class A:
  """This is an example class"""

  def meth(self, x):
    """This is an example method that takes one parame-
ter other than self
    and returns a dictionary"""

    return {}
```

myfile2.py:

```
import myfile1
a = myfile1.A()
a.meth|
```

In this case, after the user has typed `a.meth`, Wing will look up the docstring and call signature for the `meth` method on class `A` and will display information about it in the source assistant:

```
File:            myfile1.py, line 4
Call Signature:  A.meth(self, x) -> dict
Doc String:      This is an example method that takes one pa-
rameter other
                 than self and returns a dictionary
```

Note that Wing cannot determine the types of all arguments or return values, but presents as much information as it can glean from the source code. To help Wing produce a more complete analysis of your source code, you can add statements such as the following to provide hints of the types of values:

```
assert isinstance(myvalue, mymodule.CMyClass)
```

For extension modules written in C/C++, Wing will try to load and inspect the module to determine its contents. However, due to the design of Python's extension module capability, the resulting information is more limited than what can be obtained from a Python file. To augment this, Wing can be given an interface file, which is a Python skeleton that replicates the functions, attributes, classes, and methods defined by the extension module. These files should be given the name of the extension module plus `*.pi`. For example, for an extension module imported as `mymodule`, the interface file would be named `mymodule.pi`. Examples of interface files can be see in `resources/builtin-pi-files` inside your Wing IDE installation.

4.11. Auto-save

The source code editor auto-saves files to disk every few seconds. The auto-save files are placed in a subdirectory of your **User Settings Directory**.

If Wing ever crashes or is killed from the outside, you can use these files to recover any unsaved changes. Copy the auto-save files to overwrite the older unsaved files, doing a comparison first to verify that the auto-save file is what you want.

4.12. Notes on Copy/Paste

There are a number of ways to copy and paste text in the editor:

- Use the Edit menu items. This stores the copy/cut text in the system-wide clipboard and can be pasted into or copied from other applications.
- Use key equivalents as defined in the Edit menu.
- Right-click on the editor surface and use the items in the popup menu that appears.
- Select a range of text and drag it using the drag and drop feature (pressing shift before dropping will move text rather than copy it)
- On Linux, select text anywhere on the display and then click with the middle mouse button to insert it at the point of click.
- In emacs mode, ctrl-k (`kill-line`) will cut one line at a time into the private emacs clipboard. This is kept separate from the system-wide clipboard and is pasted using ctrl-y (`yank-line`). On Windows and Mac OS X, ctrl-y will paste the contents of the system-wide clipboard only if the emacs clipboard is empty.

- In VI mode, named text registers are supported.
- On Windows and Mac OS X, click with the middle mouse button to insert the current emacs private clipboard (if in emacs mode and the buffer is non-empty) or the contents of the system-wide clipboard (in all other cases). On Mac OS X, the middle mouse button is emulated by holding down the command key while clicking.

It's important to note which actions use the system-wide clipboard, which use the emacs private clipboard or VI registers, and which use the X windows selection (X Windows only). Otherwise, these commands are interchangeable in their effects.

4.13. Auto-reloading Changed Files

Wing's editor detects when files have been changed outside of the IDE and can reload files automatically, or after prompting you for permission. This is useful when working with an external editor, or when using code generation tools that rewrite files.

Wing's default behavior is to automatically reload externally changed files that you have not yet been changed within Wing's source editor, and to prompt to reload files that have also been changed in the IDE.

You can change these behaviors by setting the value of the **Reload when Unchanged** and **Reload when Changed** preferences

On Windows, Wing uses a signal from the OS to detect changes so notification or reload is usually instant. On Linux and Unix, Wing polls the disk by default every 3 seconds; this frequency can be changed with the **External Check Freq** preference.

4.14. Search/Replace

Wing provides a number of tools for search and replace in your source code. Which you use depends on the complexity of your search or replace task and what style of searching you are most familiar with.

4.14.1. Toolbar Quick Search

One way to do simple searches is to enter text in the search area of the toolbar. This scrolls as you type to the next match found after the current cursor position. Pressing

`Enter` will search for each subsequent match, wrapping the search when the end of the file is reached.

Text matching during toolbar quick search is case-insensitive unless you enter a capital letter as part of your search string.

If focus is off the toolbar search area and it already contains a search string, clicking on it will immediately start searching in the current source editor for the next match. If you wish to search for another string instead, delete the text and type the desired search string. As you delete, the match position in the editor will proceed backward until it reaches your original search start position, so that after typing your new search string you will be presented with the first match after the original source editor cursor position.

4.14.2. Keyboard-driven Mini-Search/Replace

The Edit menu contains a Mini-Search sub-menu that enumerates the available keyboard-driven search options. These are normally initiated with the keyboard command sequences shown in the menu and can be controlled entirely by using the keyboard. All interaction with the mini-search manager occurs using data entry areas displayed on demand at the bottom of the IDE window.

The implementation of the mini-search manager is very close to the most commonly used search and replace features found in Emacs, but it is available whether or not the Emacs editor personality is being used.

The following search and replace features are available in this facility:

- **Forward** and **Backward** -- These display a search string entry area at the bottom of the IDE window and interactively search forward or backward in the current source editor, starting from the current cursor position. The search takes place as you type and can be aborted with `Esc` or `Ctrl-G`, which returns the editor to its original cursor location and scroll position.

  Searching is case-insensitive unless you enter a capital letter as part of your search string. To search repeatedly, press `Ctrl-U` (`Ctrl-S in emacs keyboard mode) to search forward and ‘‘Ctrl-Shift-U` (`Ctrl-R` in emacs mode) to search in reverse. The search direction can be changed any number of times and searching will wrap whenever the top or bottom of the file is reached. You can also enter `Ctrl-U` (`Ctrl-S` in emacs mode) or `Ctrl-Shift-U` (`Ctrl-R` in emacs mode) initially while the search string is still blank in order to call up the most recently used search string and begin searching forward or backward with it.

- **Selection Forward** and **Selection Backward** -- These work like the above but start with the selection in the current source editor.

- **Regex Forward** and **Regex Backward** -- These work like the above but treat the search string as a regular expression.
- **Query/Replace** and **Regex Query/Replace** -- This prompts for search and replace strings in an entry area at the bottom of the IDE window and prompts for replace on each individual match found after the cursor location in the current source editor. Press `y` to replace or `n` to skip a match and move on to the next one. The interaction can be canceled at any time with `Esc` or `-G`. Matching is case insensitive unless a capital letter is entered as part of the search string. Searching is always forward and stops when the end of the file is reached, without wrapping to any un-searched parts between the top of the file and the position from which the search was started.
- **Replace String** and **Replace Regex** -- This works like the above command but immediately replaces all matches without prompting.

4.14.3. Search Tool

The dockable `Search` tool can be used for more advanced search and replace tasks on the current editor. It provides the ability to customize case sensitivity and whole/part word matching, search in selection, and perform wildcard or regex search and replace.

To the right of the `Search` and `Replace` fields, Wing makes available a popup that contains a history of previously used strings, options for inserting special characters, and an option for expanding the size of the entry area.

The following search options can be selected from the tool:

- **Case Sensitive** -- Check this option to show only exact matches of upper and lower case letters in the search string.
- **Whole Words** -- Check this option to require that matches are surrounded by white space (spaces, tabs, or line ends).
- **In Selection** -- Search for matches only within the current selection on the editor.

The following additional options are available from the Options popup menu:

- **Show Replace** -- Whether or not the Replace fields are visible on the tool.
- **Text Search** -- Select this to do a regular text search without wildcard or regex.
- **Wildcard Search** -- Select this to allow use of special characters for wildcarding in the search string (see below for details).

- **Regex Search** -- Select this to use regular expression style searching. This is a more powerful variant than wildcard search that allows for more complex specification of search matches and replacement values. For information on the syntax allowed for the search and replace strings, see Python's Regular Expression Syntax documentation.
- **Wrap Search** -- Uncheck this to avoid wrapping around when the search reaches the top or bottom of a file.
- **Incremental** -- Check this to immediately start or restarted searching as you type or alter search options. When unchecked, use the forward/backward search buttons to initiate searching.
- **Find After Replace** -- Select this to automatically find the next search match after each Replace operation.

Wildcard Search Syntax

`*` can be used to match any sequence of characters except for line endings. For example, the search string `my*value` would match anything within a single line of text starting with `my` and ending with `value`. Note that `*` is "greedy" in that `myinstancevalue = myothervalue` would match as a whole rather than as two matches. To avoid this, use Regex Search instead with `.*?` instead of `*`.

`?` can be used to match any single character except for line endings. For example, `my???value` would match any string starting with `my` followed by three characters, and ending with `value`.

`[` and `]` can be used to indicate sets of match characters. For example `[abcd]` matches any one of `a`, `b`, `c`, or `d`. Also, `[a-zA-Z]` matches any letter in the range from `a` to `z` (inclusive), either lower case or uppercase. Note that case specifications in character ranges will be ignored unless the `Case Sensitive` option described above is turned on.

4.14.4. Search in Files Tool

The dockable `Search in Files` tools is used to search and replace within sets of files, or for searching Wing's documentation. It performs searches in batch and displays a result list for all found matches.

Searching may span the current editor, a single selected file, all open files, all project files, all of Wing's documentation, or sets of files on disk.

Files in a set may be filtered by file type, for example searching only through Python files in the project.

In addition the options also available in the `Search` tool, the following choices are available in the Options popup menu:

- **Replace Operates On Disk** -- Check this to replace text in unopened files directly on disk. See **Replace in Multiple Files** for details on this option.

- **Recursive Directory Search** -- Check this to search recursively within all subdirectories of the selected search directory.
- **Omit Binary Files** -- Check this to omit any file that appears to contain binary data.
- **Auto-restart Searches** -- Check this to restart searching immediately if it is interupted because a search parameter or the set of files being searched is changed.
- **Open First Match** -- Check this to automatically open the first batch search match, even before the result list is clicked upon.
- **Show Line Numbers** -- Check this to include line numbers in the batch result area.
- **Result File Name** -- This is used to select the format of the result file name shown in the batch result area.

4.14.4.1. Replace in Multiple Files

For searches that operate on open files, replace always occurs in the open file editor and can be undone or saved to disk subsequently, as with any other edit operation.

When replacing text in batch mode, some of the files being searched may not currently be open in an editor. In this case, Wing will by default open all altered files and make changes in newly created editors that remain open until the user saves and closes them explicitly. This is the safest way to undertake multi-file global replace operations because it clearly shows which files have been altered and makes it possible to undo changes.

An alternative approach is available by Selecting the Replace Operates on Disk option from the Options popup. This will cause Wing to change files directly on disk in cases when there is no currently open editor.

Because global replace operations can be tricky to do correctly, we strongly recommend using a revision control system or frequent backups and manually comparing file revisions before accepting files that have been altered.

4.15. User-defined Bookmarks

Wing IDE Professional and higher support named user-defined bookmarks that can be set and accessed from the Source menu and the key bindings shown there. Defined marks are listed in the **Bookmarks** tool. Bookmark names are global to the project and refer to a particular position within a selected file:

- **For Python files,** bookmarks are defined relative to the enclosing scope (method, class, or function), so edits before the line where the bookmark is located will usually not cause the bookmark's relative position in source code to be changed. Only edits between the anchoring scope, such as start of method, and the bookmarked line will cause a bookmark's position to slip. Wing currently does not try to track bookmarks when this is the case, but they can easily be redefined if exact location is important.
- **For all other types of files,** bookmarks are defined simply by file name and line number. If the file is edited, the bookmark's position may appear to slip.

When navigating to a bookmark from the Source menu or key binding, Wing will present a dialog or entry area at bottom of the screen (depending on editor personality) into which the bookmark name can be typed. A list of possible completions will be displayed. Pressing tab will select the currently highlighted completion.

A list of defined bookmarks is available in the Bookmarks tool, which is available from the Tools menu. Right click on an entry for a context menu of operations for the selected bookmark or bookmarks. Multi-selection is possible by holding down the shift and/or control keys. Double clicking or middle mouse clicking will navigate to the selected bookmark.

When the Bookmarks tool has focus, keyboard navigation is possible with the arrow keys and by typing letters to move quickly to a particular bookmark. Enter can then be pressed to navigate to the selected bookmark.

In VI mode, the standard `m` and `\‘` plus key bindings are supported, in addition to the operations in the Source menu, which allow for the definition of bookmarks with names longer than one character.

Emacs, Brief, and other key bindings also support bookmarks. However, bookmark functionality for VI, Emacs, and Brief key bindings is omitted in Wing IDE Personal.

4.16. Templating (Code Snippets)

Wing provides support for defining and using templates for commonly reused bits of code (sometimes called code snippets) and other text. Templates might be used for standard file skeletons, comment formats, dividers, class definitions, function definitions, HTML tables, and much more. Although Wing comes with a few example templates, in most cases users will want to define their own templates, to match their coding conventions and preferences.

Wing's templating facility is implemented using the **scripting sub-system** and is controlled from the **Templating** tool panel. In most cases, key bindings are assigned to templates so that they templating tool does not have to be visible in order to use a template.

Overview

Templates are located either in `scripts/templates` inside the Wing IDE installation or in templates in the **user settings directory**. When a template of the same name is found in both, the template in the user settings directory will be used in preference whenever the template is referred to by name (for example, when assigning key bindings or invoking it with the `template` or `template-file` commands).

Advanced users can add additional template directories by altering `gTemplateDirs` global in the `templating.py` script.

Each template is in a file with name in the form `name.ext` where `name` is the name of the template and `ext` is the file extension that should be used when using the template to create a new file.

Syntax

Templates are text that contains markers where user-provided values should be inserted. These markers are similar to Python's `%(varname)s` string substitution syntax but instead of containing only a variable name, the body of the marker contains richer argument collection information in the following format, with vertical bars dividing each value:

```
%(varname|type|default)s
```

Type and default are optional but the vertical bar must be present if omitting type but including a default. To write a template that includes Python style string formats, escape each `%` by writing `%%` instead.

Each part is defined as follows:

varname -- The name of the variable. When the value is collected from the user, underscores will be replaced by spaces and the words capitalized. For example, "user_name" will be rendered "User Name".

Any number of the following special characters may be prefixed to the variable name to control how it is used:

Exclamation point (!) indicates that the value should be shown for data collection even if a default value can be found for it. Otherwise, it is hidden when a default is found.

At sign (@) indicates that the value should be wrapped if it exceeds the configured **text wrap line column**.

type -- The type of data to collect. Currently this is one of:

string(length) -- a string with given maximum length (uses default 80 chars if length is omitted)

filename -- a file name

date -- current date in locale's preferred format or in the `time.strftime()` format given in the environment variable `__DATE_FORMAT__`

datetime -- current date+time in locale's preferred format or in the `time.strftime()` format given in the environment variable `__DATETIME_FORMAT__`

If this field is omitted or empty, string is assumed.

default -- The default value to use. This may be the actual value, or may contain environment variable references in the form `$(envname)` to attempt to read all or part of the value from the named environment variable.

Environment variables can be specified either in the `Debug` tab of Wing's **Project Properties** or in the environment that exists before Wing is launched. Values in the Project Properties override any values set before starting Wing.

When this field is omitted, or when no default environment value can be found, the user will be prompted to enter the value.

Indentation and Line Endings

Templates should always use one tab for each level of indentation. Tabs will be replaced with the appropriate indentation type and size when the template is used in a new or existing file (either according to content of the target file or using the configured **indent**

style and **indent size** for new files). Wing will force tab indentation in all newly created template files.

Similarly, line endings in templates will be replaced with the appropriate type to match the file to which the template is applied. However, there is no requirement for template files to contain any particular kind of line ending.

If the template starts with `|x|` then `x` is a specification of how the indents in the template should be converted. It can be one of:

- *An integer*: Re-indent as a block, like Wing's indent-region command, so the first line is at the given number of indent levels.
- *The character 'm'*: Re-indent as a block, like Wing's indent-to-match command, so the first line is at the expected indent level according to context in the source.
- *The character 'm' followed by '+' or '-' and an integer*: Re-indent as for `'m'` and then shift left or right by the given number of indents.

Any `|x|` at the start of a template file will be removed before the template is inserted into an editor.

Cursor Placement

Templates can contain `|!|` to indicate where the cursor should be placed once the template has been inserted into a file. This mark will be removed before templates are inserted into an editor.

Reloading

The templating script will reload templates whenever they change on disk and will print warnings about any that cannot be parsed into the `Scripts` channel of the **Messages** tool.

Commands

Once the templating support script has loaded into Wing, the following commands will be available for invoking templates:

template -- This will insert a template (selected by name) at the cursor in the current editor. If there is a non-empty selection on the editor, it will replace the selection. The

user will be prompted for any arguments defined by the template, if they cannot be found in defaults.

template-file -- This will create a new file of the type specified by the template file's extension and insert the selected template into it, prompting the user as needed for arguments.

User Interface

When templates are executed, Wing will prompt for any missing arguments found in the template (those for which no defaults can be determined; see above description of template syntax). Argument collection is achieved with the same built-in automatic argument collection engine that Wing uses to obtain missing command arguments, usually at the bottom of the current editor window.

The templating script also registers a new tool type with Wing IDE. The tool is not shown by default but can be inserted into Wing IDE dock windows from the `Tools` menu and the `Insert Tool` sub-menu of the tool area context menu. The tool panel currently supports adding, editing, removing, and executing templates, and also assigning key bindings for pasting selected templates into the current editor.

4.17. Using Revision Control with Wing

Wing integrates the most common revision control operations for CVS (the Concurrent Versions System), Subversion (a more modern replacement for CVS), and Perforce (a commonly used commercial solution). It can be used to:

- Add files or directories to the repository
- Update to obtain changes from the repository
- Commit edits to the repository
- View differences between local copies and the repository or between most recent and previous repository version
- Obtain revision log and status information
- Revert a file to the repository version
- Under Subversion, list, blame/praise, and resolved are also supported

To turn on revision control for a project, use the **Extensions** tab in `Project Properties` and select the desired revision control system. A new menu `CVS`, `SVN`, or `Perforce`

will appear in the menu bar and operations will be added to the editor and project manager context menus.

This integration assumes that you have already set up CVS, Subversion, or Perforce to work on your system and already have a repository checked out on your disk. If not, refer to the notes below.

Please send us suggestions, comments, or requests using the **Feedback** feature in the `Help` menu or by emailing to support at wingware dot com

Installing CVS

On Windows: Download from http://www.nongnu.org/cvs and add installation location to `PATH` environment variable from the Advanced tab of the System control panel

On Linux/Unix: Install CVS from using the packages that came with your Linux/Unix distribution or download from http://www.nongnu.org/cvs and build from sources.

Installing Subversion

On Windows: Download from http://subversion.tigris.org/ and add installation location to `PATH` environment variable from the Advanced tab of the System control panel

On Linux/Unix: Install Subversion from using the packages that came with your Linux/Unix distribution or download from http://subversion.tigris.org/ and build from sources.

Installing Perforce

To install and use Perforce, please refer to the vendor's documentation. Before trying Perforce in Wing, verify that it works from the command line and you have already checked out a repository. Then **enable revision control** under the `Extensions` tab in your `Project Properties` and set **Revision Control System** to `Perforce`.

If Wing is not inheriting the `P4PORT` or `P4HOST` environment variables you may also need to set these in the `Environment` in your **Project Properties**. For example, you might add the following if running Perforce on port `1666` on a machine called `myhostname`:

```
P4PORT=1666
P4CLIENT=myhostname
```

Using SSH Repositories

Both CVS and Subversion can use SSH as a secure and convenient way to access the revision control repository.

To set up SSH on Windows:

1. Install putty -- the combined installer is easiest
2. Add the location where putty is installed to your `PATH` environment variable from the Advanced tab of the System control panel.
3. Run `puttygen` and generate an SSH2 RSA key pair. Use a passphrase you will remember. Save both private and public keys to disk. Copy the contents of the key box (starting with "ssh-rsa") to `rsa-public.key` on disk.
4. Copy the `rsa-public.key` file to your server and add it to the `.ssh/authorized_keys` file under your username. E.g., use `pscp rsa-public.key user@hostname:` and then log into hostname and `cat rsa-public.key >> .ssh/authorized_keys`.
5. Run putty and enter host name in `Host Name` and `Saved Sessions` boxes then press `Save`. Go to the `Connection` category and enter your user name on the server into the `Auto-login` username box. Go back to `Session` category and press `Save` again.
6. Run `pageant`, which adds an icon to your Windows tray. Right click and select `Add Key`. Navigate to the private key saved from `puttygen` and enter your passphrase when prompted.
7. Restart `putty`, click on the saved session, press Load, and then Open. This should open a connection to the server without prompting for any further information.

To set up SSH on Linux/Unix:

If you do not already have `openssh` and `cvs` installed, install them from packages that came with your Linux or Unix distribution.

1. If `ssh-add -l` complains that it cannot find the SSH agent, run `ssh-agent bash` (or your favorite shell). This can be skipped on most modern Linux distributions because they run the X window manager inside ssh-agent.
2. If you don't already have an ssh key in `.ssh`, issue the command `ssh-keygen -t rsa` to create a key pair in `.ssh/id_rsa` (the private key) and `.ssh/id_rsa.pub` (the public key). Enter a passphrase you will remember.

3. Copy the file `.ssh/id_rsa.pub` to your server and add it to the `.ssh/authorized_keys` file under your username. E.g., use `scp rsa-public.key user@hostname:` and then log into hostname and `cat rsa-public.key >> .ssh/authorized_keys`.

4. Back on your client (where you plan to run Wing), type `ssh-add` and enter your passphrase to get the SSH key loaded into `ssh-agent`.

5. Type `ssh user@hostname` and you should be able to log into your server without being asked for a password.

Subversion with SSH

First time configuration: Install and configure SSH as described above (this also loads authentication information into the cache for the current session)

To check out a repository: Type `svn checkout svn+ssh://hostname/path/to/repository`. If you're not sure what to check out try this first: `svn list svn+ssh://hostname/`

Future sessions require: On Windows, double click on your private key file and enter your pass phrase, or **On Linux/Unix**, run `ssh-add` and enter your pass phrase. Then run Wing with a project where the **Enable Revision Control** property is set in the Extensions tab of Project Properties and Subversion is selected as the revision control system.

Subversion with http/https or file URLs

To check out a repository with http or https, type `svn checkout http://hostname/path/to/repository`. If you're not sure what to check out try this first: `svn list http://hostname/`

To check out a repository with file: URLs, type `svn checkout file:///path/to/repository` You will be prompted for your user name and password, which will be cached by Subversion for future sessions.

Future sessions require: Run Wing with a project where the **Enable Revision Control** property is set in the Extensions tab of Project Properties and Subversion is selected as the revision control system.

Subversion without Authentication Cache

If you are using Subversion with http, https, or file access method and authentication cache disabled, you will need to go into the `Options` in the `SVN` menu in Wing (after enabling Subversion for your project) and select `Manual (from Wing)` for the `Authentication` option. This will prompt you for a user name and password the first time each repository is used in a Wing IDE session and sends them to Subversion with the `--username` and `--password` command line arguments, along with `no-auth-cache` so that your entries are never cached on disk.

Using CVS with SSH

First time configuration: Install and configure SSH as described above (this also loads authentication information into the cache for the current session). **On Windows**, add `CVS_RSH=plink` to your environment from the Advanced tab of the System control panel. **On Linux/Unix**, add `CVS_RSH=ssh` to your environment. For example, `CVS_RSH=ssh; export CVS_RSH` on the command line, or add this to your `.bashrc` file. Note that **Environment** in your Project Properties can also be used to set `CVS_RSH` or other environment variables, however only for CVS commands issued from the IDE.

To check out a repository: Type `cvs -d :ext:username@hostname:/path/to/repository co module_name`

Future sessions require: On Windows, double click on your private key file and enter your pass phrase, or **On Linux/Unix**, run `ssh-add` and enter your pass phrase Then run Wing with a project where the **Enable Revision Control** property is set in the Extensions tab of Project Properties and CVS is selected as the revision control system.

Using CVS with pserver

CVS's pserver authentication mechanism is obsolete but it is still used for anonymous CVS access in some places, such as on sourceforge.net. If you are working with a pserver repository that requires a password (Sourceforge does not), then you will need to issue `cvs login` once from the command line before starting Wing.

Notes on the Implementation

Wing's revision control integrations are based on the IDE's scripting extension API. Additional revision control systems can be added by basing on the `cvs.py`, `svn.py`, or `perforce.py` sources found in the `scripts` directory within the Wing IDE installation.

If you plan to work on scripts that are in the `scripts` directory, copy them first to your **User Settings Directory** in the `scripts` directory there. When duplicate script names are found Wing will prefer those found in your user settings directory, so this allows you to make changes without losing those changes when Wing is updated in the future.

For more information on scripting, see **Scripting and Extending Wing IDE**.

4.18. Keyboard Macros

The Edit menu contains items for starting and completing definition of a keyboard or command sequence macro, and for executing the most recently defined macro. Once macro recording is started, any keystroke or editor command is recorded as part of that macro, until macro recording is stopped again. Most commands may be included in macros, as well as all character insertions and deletions.

4.19. Source Code Analysis

Wing's auto-completer, source index menu, goto-definition capability, some of the source reformatting features, and in Wing IDE Professional the source code browser and source assistant all rely on a central engine that reads and analyzes your source code in the background as you add files to your project or alter your code in the source code editor.

How Analysis Works

In analysing your source, Wing will use the Python interpreter and `PYTHONPATH` that you have specified in your **Project Properties**. If you have indicated a main debug file for your project, the values from that file's properties are used; otherwise the project-wide values are used. Whenever any of these values changes, Wing will completely re-analyze your source code from scratch.

You can view the Python interpreter and `PYTHONPATH` that are being used by the source code analysis engine, by selecting the Show Analysis Stats item in the Source menu. The values shown in the resulting dialog window are read-only but may be changed by

pushing the Settings button. See **Project-wide Properties** for details on changing these values.

Be aware that if you use multiple versions of the Python interpreter or different `PYTHONPATH` values for different source files in your project, Wing will analyse all files in the project using the one interpreter version and `PYTHONPATH` it finds through the main debug file or project-wide debug properties settings. This may lead to incorrect or incomplete analysis of some source, so it is best to use only one version of Python with each Wing IDE project file.

When Wing tries to find analysis information for a particular module or file, it takes the following steps:

- The path and same directory as the referencing module are searched for an importable module
- If the module is Python code, Wing statically analyses the code to extract information from it
- If the module is an extension module, Wing looks for a *.pi interface description file as described later in this section

  Inline emphasis start-string without end-string.
- If the module cannot be found, Wing tries to import it in a separate process space in order to analyze its contents

Limitations

The following are known limitations affecting features based on source analysis:

- Argument number, name, and type is not determined for functions and methods in extension modules
- Analysis sometimes fails to identify the type of a construct because Python code doesn't always provide clues to determine the data type. In these cases, you may use `isinstance` and/or interface files to inform the analyzer, as described below.
- Types of elements in lists, tuples, and dictionaries are not identified.
- Analysis information may be out of date if you edit a file externally with another editor and don't reload it in Wing. See section **Auto-reloading Changed Files** for reload options.
- Some newer Python language constructs and possible type inferencing cases are not explicitly supported.

Using isinstance() to Assist Analysis

One way to inform the code analysis facility of the type of a variable is to add an `isinstance` call in your code. An example is `assert isinstance(obj, CMyClass)`. The code analyzer will pick up on these and present more complete information for the asserted values.

Using *.pi files to Assist Analysis

Wing's source analyser can only read Python code and does not contain support for understanding C/C++ extension module code other than by attempting to import the extension module and introspecting its contents (which yields only a limited amount of information and cannot determine argument number, name, or types).

To inform the code analysis facility of the contents of an extension module, it is possible to create a `*.pi` (Python interface) file. For example, for a module imported as `mymodule`, the interface file is called `mymodule.pi`. This file is simply a Python skeleton with the appropriate structure and call signature to match the functions, attributes, classes, and methods defined in an extension module. In many cases, these files can be auto-generated from interface files.

Wing will search for `*.pi` files first in the same directory as it finds the extension module (or its source code if it has not yet been compiled and the source code's directory is on your configured Python Path), If not found, Wing will look in the directory path set with the **Interfaces Path** preference. Next, Wing will look in the `resources/builtin-pi-files` directory within your Wing IDE installation. Finally, Wing will look in `resources/packages-pi-files`, which is used to ship some `*.pi` files for commonly used third party packages.

When searching on the interfaces path or in the `resources` directories, the top level of the directory is checked first for a matching `*.pi` file. Then, Wing tries looking in a sub-directory `#.#` named according to the major and minor version of Python being used with your source base, and subsequently in each lower major/minor version back to `1.5`.

For example, if `c:\share\pi\pi-files` is on the interfaces path and Python 2.3 is being used, Wing will check first in `c:\share\pi\pi-files`, then in `c:\share\pi\pi-files\2.3`. then in `c:\share\pi\pi-files\2.2`, and so forth.

Example `*.pi` files used by Wing internally to produce autocompletion information for builtins can be seen in the directory `resources/builtin-pi-files` inside your Wing IDE installation. This also illustrates the above-described version number fallback mechanism.

In cases where Wing cannot find a `*.pi` at all, it will attempt to load the module by name (in a separate process space) so that it can introspect its contents. The results of

this operation are stored in `pi-cache` within the **User Settings Directory** and used subsequently. This file is regenerated only if the `*.pyd` or `*.so` for the loaded module changes.

4.19.1. Analysis Disk Cache

The source code analyzer writes information about files it has recently examined into the Cache Directory that is specified in the About box accessible from the `Help` menu.

Cache size may be controlled with the **Max Cache Size** preference However, Wing does not perform well if the space available for the cache is smaller than the space needed for a single project's source analysis information. If you see excessive sluggishness, either increase the size of the cache or disable it entirely by setting its size to 0.

If the cache will be used by more than one computer, make sure the clocks of the two computers are synchronized. The caching mechanism uses time stamps, and may become confused if this is not done.

The analysis cache may be removed in its entirety with no ill effects.

Source Code Browser

The source code browser, which is available only in Wing IDE Professiona, acts as an index to your source code, supporting inspection of collections of Python code from either a module-oriented or class hierarchy-oriented viewpoint.

> **Background Source Analysis**
> Wing IDE's source code analyzer will run in the background from the time that you open a project until all files have been analyzed. You may notice this overhead during the first 5 to 30 seconds after you have opened your project, depending on the size of your source base. Until analysis is complete, the class-oriented view within the browser window will only include those classes that have been analyzed. This list is updated as more code is analyzed.

5.1. Display Choices

The source code browser offers three ways in which to browse your source code: All code by module, all code by class hierarchy, or only the current file by class hierarchy. These are selected from the menu at the top left of the browser.

5.1.1. Viewing by Module

Viewing by module shows in alphabetical order all Python modules that you have placed into your project and all modules reachable by traversing the directory structure that contains your project files (including all sub-directories). The following types of top-level units are used in this display mode:

- Packages are directories that contain a number of files and a special file `__init__.py`. This file optionally contains a special variable `__all__` that lists the file-level modules Python should automatically import when the package as a

whole is imported. See the Python documentation for additional information on creating packages.

- Directories found in your project that do not contain the necessary `__init__.py` file are listed as 'directory' rather than 'package' in the source browser window.
- Python files found at any level are denoted as 'module'.

Within each top-level package, directory, or module, the browser will display all sub-modules, sub-directories, modules, and any Python constructs. These are all labeled by generic type, including the following types:

- **variable** -- a variable defined at the top-level of a Python module
- **function** -- a function defined at the top-level of a Python module
- **class** -- an object class found in Python source
- **method** -- a class method
- **attribute** -- a class or instance attribute

5.1.2. Viewing Class Hierarchy

When viewing by class hierachy, the browser replaces the hierarchical tree view with a list of all classes found in analyzed code, in alphabetical order.

In this display mode, the structure on disk of your packages, directories, and modules is completely hidden from view. Instead, the hierarchy of your classes is displayed, starting at base classes and working downward to derived classes.

Within each class, in addition to a list of derived classes, the methods and attributes for the class are shown.

5.1.3. Viewing Current Module

The browser can also be asked to restrict the display to only those symbols defined in the current module. This view shows all types of symbols at the top level and allows expansion to visit symbols defined in nested scopes.

5.2. Display Filters

A number of options are available for filtering the constructs that are presented by the source code browser. These filters are available from the `Options` popup menu at the top right of the browser. They are organized into two major groups: (1) construct scope and source, and (2) construct type.

5.2.1. Filtering Scope and Source

The following distinctions are made. Constructs in each category can be shown or hidden as a group:

- **Public** -- Constructs accessible to any user of a module or instance. These are names that have zero leading underscores, such as `Print()` or `kMaxListLength`.
- **Semi-Private** -- Constructs intended for use only within related modules or from related or derived classes. These are names that have one leading underscore, such as `_NotifyError()` or `_gMaxCount`. Python doesn't enforce usage of these constructs, but they are helpful in writing clean, well-structured code and are recommended in the Python language style guide.
- **Private** -- Constructs intended to be private to a module or class. These are names that have two leading underscores, such as `__ConstructNameList()` or `__id_seed`. Python enforces local-only access to these constructs in class methods (see the Python documentation for details).
- **Inherited** -- Constructs inherited from a super-class.
- **Imported** -- Constructs imported into a module with an import statement.

5.2.2. Filtering Construct Type

Constructs in the source code browser window can also be shown or hidden on the basis of their basic type within the language:

- **Classes** -- Classes defined in Python source.
- **Methods** -- Methods defined within classes.
- **Attributes** -- Attributes (aka 'instance variables') of a class. Note that these can be either class-wide or per-instance, depending on whether they are defined within the class scope or only within methods of the class.

- **Functions** -- Non-object functions defined in Python source (usually at the top-level of a module).
- **Variables** -- Variables defined anywhere in a module, class, function, or method (but not including function or method parameters).

5.3. Sorting the Browser Display

In all the display views, the ordering of constructs within a module or class can be controlled from the `Options` popup menu in the browser.

- **Alphabetically** -- Displays all entries in the tree in alphabetic order, regardless of type.
- **By Type** -- Sorts first by construct type, and then alphabetically.
- **In File Order** -- Sorts the contents of each scope in the same order that the symbols are defined in the source file.

5.4. Navigating the Views

To navigate source code from the browser, double click on the tree display. This will open source files to the appropriate location. Note that the source files opened in this way will automatically close when browsing elsewhere, except if they are edited or if the stick pin icon in the upper right of the source area is clicked to indicate that the source file should remain open until explicitly closed by the user.

Right-clicking on classes will present a popup menu that includes any inherited classes, allowing quick traversal up the class hierarchy.

The option `Follow Selection` may be enabled in the `Options` menu to cause the browser to open files even on a single click or as the currently selected item on the browser is changed from the keyboard.

5.5. Browser Keyboard Navigation

Once it has the focus, the browser tree view is navigable with the keyboard, using the up/down arrow keys, page up and page down, home/end, and by using the right arrow key on a parent to expand it, or the left arrow key to collapse a parent.

Holding down the shift key while pressing the right arrow will expand recursively below the expansion point. Recursive expansion is capped at five additional levels for each operation to avoid difficult-to-detect infinite recursions.

Whenever a tree row is selected, pressing enter or return will open the source view for the selected symbol in a separate window, indicating the point of definition for that symbol.

Debugger

Wing's debugger provides a powerful toolset for rapidly locating and fixing bugs in Python code. It supports breakpoints, stepping through code, inspecting and changing stack or module data, watch points, expression evaluation, and command shell style interaction with the paused debug process.

The debugger is built around a TCP/IP client/server design that supports launching your application not just from Wing itself but also externally, as with CGI scripts or code running in an embedded scripting facility within a larger application. Remote (host to host) debugging is also provided.

Because the debugger core is written in optimized C, debug overhead is relatively low; however, you should expect your programs to run about 50% slower within the debugger.

6.1. Quick Start

Wing IDE can be used to debug all sorts of Python code, including scripts and stand-alone applications written with **wxPython**, Tkinter, **PyQt**, **PyGTK**, and **pygame**. Wing can also **debug web CGIs** including those running under **mod_python**, code running under **Zope** and **Plone**, and code running in an embedded Python interpreter.

This section describes how to debug stand-alone scripts and applications that can be launched from within Wing IDE. If you wish to debug web CGIs within the web server, web servlets, or embedded Python scripts, please refer to **Debugging Externally Launched Code** and **Remote Debugging**.

Before debugging, you will need to install Python on your system if you have not already done so. Python is available from www.python.org.

To debug Python code with Wing, open up the Python file and select `Start / Continue` from the Debug menu. This will run to the first breakpoint, unhandled exception, or until the debug program completes. Select `Step Into` instead to run to the first line of code.

Unexpected Exceptions while Debugging
Wing may report exceptions that you don't normally see when running your debug process. This happens when exceptions occur that are handled (or cleared) in C or C++ extension module code. Wing detects all exceptions not handled in Python code. You can check the `Ignore this exception location` checkbox in the Exceptions tool to avoid seeing repeated reports of such exceptions, if they are not of interest. This behavior can also be altered from the `Debugger / Exceptions` preference group.

Use the Debug I/O tool to view your program's output, or to enter values for input to the program. If your program depends on characteristics of the Windows Console or a particular Linux/Unix shell, see **External I/O Consoles** for more information.

In some cases, you may also need to enter a `PYTHONPATH` and other environment values using the Project Properties dialog available from the Project menu. This can also be used to specify which Python interpreter you wish to use with your debug process. Use this if Wing IDE cannot find Python on your system or if you have more than one version of Python installed.

To set breakpoints, just click on the leftmost part of the margin next to the source code. In Wing IDE Professional, conditional and ignore-counted breakpoints are also available from the Debug / Breakpoints Options menu or by right clicking on the breakpoints margin.

6.2. Specifying Main Entry Point

Normally, Wing will start debugging in whatever file you have active in the frontmost editor. Depending on the nature of your project, you may wish to specify a file as the default debug entry point.

To do this, right-click on one of your Python files in the project manager window and choose the Set As Main Debug File option from the popup menu, or use the Set Current as Main Debug File item in the Debug menu.

This file is subsequently run whenever you start the debugger, except when using Debug Current File in the Debug menu, or when right-clicking on an entry in the project manager and choosing the Debug Selected popup menu item.

Note that the path to the main debug file is highlighted in red in the project window. You may clear the default debug entry point with the popup menu's Clear Main Debug File item or the Clear Main Debug file in the Debug menu.

The main entry point defined for a project is also used by the source code analysis

engine to determine the python interpreter version and Python path to use for analysis. Thus, changing this value will cause all source files in your project to be reanalyzed from scratch. See section **Source Code Analysis** for details.

6.3. Debug Properties

In some cases, you may need to set project and per-file properties from the Project manager before you can debug your code. This is done to specify Python interpreter, `PYTHONPATH`, environment variables, parameters, start directory, and other values associated with the debug process. For details, see **Project-Wide Properties** and **Per-file Properties**.

6.4. Setting Breakpoints

Breakpoints can be set on source code by opening the source file and clicking on the breakpoint margin to the left of a line of source code. Right-clicking on the breakpoint margin will display a context menu with additional breakpoint operations and options. In Wing IDE Professional, the `Breakpoints` tool in the `Tools` menu can be used to view, modify, or remove defined breakpoints.

Alternatively, the `Debug` menu or the toolbar's breakpoint icons can be used to set or clear breakpoints at the current line of source (where the insertion cursor or selection is located).

Breakpoint Types

In Wing IDE Professional, the following types of breakpoints are available:

- **Regular** -- A regular breakpoint will always cause the debugger to stop on a given line of code, whenever that code is reached.
- **Conditional** -- A conditional breakpoint contains an expression that is evaluated each time the breakpoint is reached. The debugger will stop only if the conditional evaluates to `true` (any non-zero, non-empty, non-`None` value, as defined by Python). You may edit the condition of any existing breakpoint with the `Edit Breakpoint Condition...` item in the Breakpoint Options group of the Debug menu, by right clicking on the breakpoint, or in the `Breakpoints` tool.

- **Temporary** -- A temporary breakpoint will be removed automatically after the first time it is encountered. No record of the breakpoint is retained for future debug runs.

Breakpoint Attributes

Once breakpoints have been defined, you can operate on them in a number of ways to alter their behavior. These operations are available as menu items in the Debug menu, in the breakpoint margin's context menu, and from the `Breakpoints` tool:

- **Ignore Count** -- It is possible to set an ignore count for a breakpoint. In this case, the breakpoint will be ignored the given number of times, and the debugger will only stop at the breakpoint if it is encountered more than the set number of times. The ignore count is reset to its original value with each new debug run. Use the `Breakpoint` tool to monitor the remaining number of times a breakpoint will be ignored.
- **Disable/Enable** -- Breakpoints can be temporarily disabled and subsequently re-enabled. Any disabled breakpoint will be ignored until re-enabled.

Breakpoints Tool

The `Breakpoints` tool, available in the `Tools` menu displays a list of all currently defined breakpoints. The following columns of data are provided:

- `Enabled` -- Checked if the breakpoint is enabled. The checkbox can be used to alter the breakpoint's state.
- `Location` -- The file and line number where the breakpoint is located
- `Condition` -- The conditional that must be true for the breakpoint to cause the debug process to stop (or blank if the breakpoint is not conditional). This value can be changed by clicking on it and editing it directly on the list.
- `Temporary` -- Checked if the breakpoint is a temporary (one-time) breakpoint. The checkbox can be used to alter the breakpoint's type.
- `Ignores` -- The number of times the breakpoint should be ignored before it causes the debugger to stop. This value can be changed by clicking on it and editing it directly on the list.

- `Ignores Left` -- The number of ignores left for a breakpoint, if a debug process exists.
- `Hits` -- The number of times the breakpoint has been reached in the current debug run (if any).

To visit the file and line number where a breakpoint is located, double click on it in the list or select `Show Breakpoint` from the context menu obtained by right-clicking on the surface of the `Breakpoints` tool. Additional options are also available from this context menu.

Keyboard Modifiers for Breakpoint Margin

Clicking on the breakpoint margin will toggle to insert a regular breakpoint or remove an existing breakpoint. You can also shift-click to insert a conditional breakpoint, and control-click to insert a breakpoint and set an ignore count for it. When a breakpoint is already found on the line, shift-click will disable or enable it, control-click will set its ignore count, and shift-control-click will set or edit the breakpoint conditional.

6.5. Starting Debug

There are several ways in which to start a debug session from within Wing:

- Choose `Start / Continue` from the Debug menu or push the `Debug` icon in the toolbar. This will run the main debug file if one has been defined (described in **Setting a Main Debug File**), or otherwise the file open in the frontmost editor window. Execution stops at the first breakpoint or exception, or upon program completion.

- Choose `Step Into` from the Debug menu or push the `Step Into` icon in the toolbar. This will run the main debug file if one has been defined, or otherwise the file open in the frontmost editor window. Execution stops at the first line of code.
- Choose `Debug Current File` from the Debug menu or `Debug Selected` from the right-click popup menu on the Project tool to run a specific file regardless of whether a main debug file has been specified for your project. This will stop on the first breakpoint or exception, or upon program completion.
- Choose `Run to Cursor` from the Debug menu. This will run the main debug file if one has been defined or otherwise the file open in the frontmost editor

window. Execution continues until it reaches the line selected in the current source text window, until a breakpoint or exception is encountered, or until program completion.

- Use `Debug Recent` in the Debug menu to select a recently debugged file. This will stop on the first breakpoint or exception, or upon program completion.
- Use one of the key bindings given in the Debug menu.

Additional options exist for initiating a debug session from outside of Wing and for attaching to an already-running process. These are described in sections **Debugging Externally Launched Code** and **Attaching**.

Once a debug process has been started, the status indicator in the lower left of the window should change from white or grey to another color, as described in **Debugger Status**.

Non-Standard Python Interpreters
If you are attempting to run your debug process against a non-standard version of Python, for example one that has been compiled with altered values for `Py_TRACE_REFS` or `WITH_CYCLE_GC`, or that has been altered in other ways, you may need to recompile the debugger core module. Please refer to section **Compiling the Wing IDE Debugger from Source** for more information. This is only possible with Wing IDE Professional, since Wing IDE Personal does not include access to the source code.

6.6. Debugger Status

The debugger status indicator in the lower left of editor Windows is used to display the state of the debugger. Mousing over the bug icon shows expanded debugger status information in a tool tip. The color of the bug icon summarizes the status of the debug process, as follows:

- **White** -- There is no debug process, but Wing is listening for a connection from an externally launched process.
- **Gray** -- There is no debug process and Wing is not allowing any external process to attach.
- **Green** -- The debug process is running.
- **Yellow** -- The debug process is paused or stopped at a breakpoint.

- **Red** -- The debug process is stopped at an exception.

The current debugger status is also appended to the Debugger status group in the IDE's Messages tool.

6.7. Flow Control

Once the debugger is running, the following commands are available for controlling further execution of the debug program from Wing. These are accessible from the tool bar and the Debug menu:

- At any time, a freely running debug program can be paused with the `Pause` item in the Debug menu or with the pause tool bar button. This will stop at the current point of execution of the debug program.
- At any time during a debug session, the `Stop Debugging` menu item or tool

Bullet list ends without a blank line; unexpected unindent.

can be used to force termination of the debug program. This option is disabled by default if the current process was launched outside of Wing. It may be enabled for all local processes by using the **Enable Kill External** preference.

When stopped on a given line of code, execution can be controlled as follows from the Debug menu or tool bar:

Step Over will step over a single line of Python code.

Step Into will attempt to step into the next executed function on the current line of code. If there is no function or method to step into, this command acts like Step Over.

Step Out will complete execution of the current function or method and stop on the first instruction encountered after returning from the current function or method.

Continue will continue execution until the next breakpoint, exception, or program termination

Run To Cursor will run to the location of the cursor in the frontmost editor, or to the next breakpoint, exception, or program termination.

Attach and **Detach** (only in Wing IDE Professional) may be used to change the debugger between different debug processes. This is for advanced users and is detailed in **Attaching and Detaching**.

6.8. Viewing the Stack

Whenever the debug program is paused at a breakpoint or during manual stepping, the current stack is displayed in the `Call Stack` tool. This shows all program stack frames encountered between invocation of the program and the current run position. Outermost stack frames are higher up on the list.

When the debugger steps or stops at a breakpoint or exception, it selects the innermost stack frame by default. In order to visit other stack frames further up or down the stack, select them in the `Call Stack` tool. You may also change stack frames using the stack selector popup menus in the `Stack Data` tool, the `Up Stack` and `Down Stack` items in the `Debug` menu, or the up/down tool bar icons.

When you change stack frames, the `Stack Data` views will be changed accordingly, and the current line of code at that stack frame is presented in an editor window.

In Wing IDE Professional, the current stack frame is also used to control evaluation context in the `Debug Probe` and `Watch` tools.

To change the type of stack display, click on the `Call Stack` tool and select from the options for display and positioning of the code line excerpted from the debug process.

When an exception has occurred, a backtrace is also captured by the `Exceptions` notification tool, where it can be accessed even after the debug process has exited.

6.9. Viewing Debug Data

The Wing IDE debugger provides several ways in which to look at your debug program's data:

(1) By inspecting locals and globals using the Stack Data tool. This area displays values for the currently selected stack frame.

(2) By browsing values in all loaded modules (as determined by `sys.modules`), using the Modules tool.

(3) By watching specific values from either of the above views (right click on values to add them to the Watch tool)

(4) By typing expressions in the Watch tool.

Values Fetched on Demand
The variable data displayed by Wing is fetched from the debug server on the fly as you navigate. Because of this, you may experience a brief delay when a change in an expansion or stack frame results in a large data transfer.
For the same reason, leaving large amounts of debug data visible on screen may slow down stepping through code.

6.9.1. Stack Data View

The Stack Data debugger tool contains a popup menu for accessing the current debug stack, a tree view area for browsing variable data in locals and globals, and a textual view area for inspecting large data values that are truncated on the tree display.

Simple values, such as strings and numbers, and values with a short string representation, will be displayed in the value column of the tree view area.

Strings are always contained in `""` (double quotes). Any value outside of quotes is a number or internally defined constant such as `None` or `Ellipsis`.

Integers can be displayed as decimal, hexadecimal, or octal, as controlled by the **Integer Display Mode** preference.

Complex values, such as instances, lists, and dictionaries, will be presented with an angle-bracketed type and memory address (for example, `<dict 0x80ce388>`) and can be expanded by clicking on the expansion indicator in the `Variable` column. The memory address uniquely identifies the construct. If you see the same address in two places, you are looking at two object references to the same instance.

Class-scoped values seen within an instance are shown in italics.

Upon expansion of complex views, the position or name of each sub-entry will be displayed in the Variable column, and the value of each entry (possibly also complex values) will be displayed in the Value column. Nested complex values can be expanded indefinitely, even if this results in the traversal of cycles of object references.

Once you expand an entry, the debugger will continue to present that entry expanded, even after you step further or restart the debug session. Expansion state is saved for the duration of your Wing IDE session.

When the debugger encounters a long string, this is indicated in the Value column by prepending `...` after the truncated string. In these cases, the full value of the string can be viewed in the textual display area at the bottom of the Stack Data tool, after clicking on the truncated string in the tree view.

Opaque Data
Some data types, such as those defined only within C/C++ code, or those containing certain Python language internals, cannot be transferred over the network. These are denoted with Value entries in the form `<opaque 0x80ce784>` and cannot be expanded further. In Wing IDE Professional you may be able to use the **Debug Probe** to access them (for example try typing `dir(value)`).

6.9.1.1. Popup Menu Options

Right-clicking on the surface of the stack data view displays a popup menu with options for navigating data structures:

- **Show/Hide Detail** -- Used to quickly show and hide the split where Wing shows expanded copies of values that are truncated on the main debug data view (click on items to show their expanded form).
- **Expand More** -- When a complex data value is selected, this menu item will expand one additional level in the complex value. Since this expands a potentially large number of values, you may experience a delay before the operation completes.
- **Collapse More** -- When a complex data value is selected, this menu item will collapse its display by one additional level.
- **Watch by ...** -- These items can be used to watch a debug data value over time, as described in **Tracking Values**.
- **Force Reload** -- This forces Wing IDE to reload the displayed value from the debug process. This is useful in cases where Wing is showing an evaluation error or when the debug program contains instances that implement `__repr__` or similar special methods in a way that causes the value to change when subjected to repeated evaluation.

6.9.1.2. Filtering Value Display

There are a number of ways in which the variable displays can be configured:

- Wing lets you prune the variable display area by omitting all values by type, and variables or dictionary keys by name. This is done by setting the two preferences, **Omit Types** and **Omit Names**.

- You can also tell Wing to avoid probing certain values by data type. This is useful to avoid attempting expansion of data values defined in buggy extension modules, which can lead to crashing of the debug process as the debugger exersizes code that isn't normally executed. To add values to avoid, set preference **Do Not Expand**.

- Wing provides control over size thresholds above which values are considered too large to move from the debug process into the variable display area. Values found to be too large are annotated as `huge` in the variable display area and cannot be expanded further. The data size thresholds are controlled with preferences **Huge List Threshold** and **Huge String Threshold**.

- By default Wing will display small items on a single line in the variable display areas, even if they are complex types like lists and maps. The size threshold used for this is controlled with preference **Line Threshold**. If you want all values to be shown uniformly, this preference should be set to `0`.

6.9.2. Tracking Values

Wing can watch debug data values using a variety of techniques for tracking the value over time. In most cases, watching a value is initiated by right-clicking a value within a tree view and selecting one of the Watch menu items. The value is then added to the list in the Watch tool and tracked by one of the following methods:

- **By Symbolic Path** - The debugger looks at the symbolic path from `locals()` or `globals()` for the currently selected stack frame, and tries to reevaluate that path whenever the value may have changed. For example, if you define a dictionary variable called `testdict` in a function and set a value `testdict[1] = 'test'`, the watched value for `testdict[1]` would show any value for that slot of `testdict`, even if you delete `testdict` and recreate it. In other words, value tracking is independent of the life of any object instances in the data path.

- **By Direct Object Reference** - The debugger uses the object reference to the selected value to track it. If you use this mode with `testdict` as a whole, it would track the contents of that dictionary as long as it exists. If you were to reassign the variable `testdict` to another value, your zoomed out display would still show the contents of the original dictionary instance (if it still exists), rather than the new value of the variable `testdict`. In other words, the symbolic path to the value is completely disregarded and only instance identity is used to track the value. Because it's meaningless to track immutable types this way, this option is disabled or enabled according to the values you select to zoom out into a separate window.

- **By Parent Reference and Slot** - The debugger uses the object reference to the parent of the selected data slot and uses a symbolic representation of the slot within the parent in order to determine where to look for any value updates. This means that reassignment of the variable that points to the parent does not alter what is displayed in the zoomed-out view; only reassignment of the selected slot changes what is displayed by the debugger.
- **By Module Slot** - This is only available for values within a module, such as `string`, `sys.path`, or `os.environ`. The debugger uses the module name to look up the module in `sys.modules` and references the value by symbolic path. Any change in the value, even across module reloads, is reflected in the Watch view.

For any of these, if the value cannot be evaluated because it does not exist, the debugger displays `<undefined>`. This happens when the last object reference to a reference-tracked value is discarded, or if a selected symbolic path is undefined or cannot be evaluated.

The Watch tool will remember watch points across debug sessions, except those that make use of an object reference, which do not survive the debug process.

6.9.3. Evaluating Expressions

The debugger Watch tool can also be used to view the value of keyboard-entered expressions. These may be entered by clicking on any cell in the Watch manager's display tree and editing or entering the desired expression in the Variable column. Press enter to complete the editing session.

Only expressions that evaluate to a value may be entered. Other statements, like variable assignments, import statements, and language constructs are rejected with an error. These may only be executed using the **Debug Probe**.

Expressions are evaluated in the context of the current debug stack frame, so this feature is available only when the debug program has been paused or has stopped at a breakpoint or exception. This also means that the value of the same typed expression may change as you move up and down the call stack in the main debugger window.

In cases where evaluating an expression results in changing the value of local or global variables, your debug program will continue in that changed context. Whenever a value is changed as a result of expression evaluation, the updated value will be propagated into any visible debugger variable display areas because Wing IDE refetches all displayed data values after the evaluation of each expression. However, since you may not notice these changes, caution is required to avoid undesired side-effects in the debug process.

Note that breakpoints are never reached as a result of expression evaluation, and any

exceptions encountered are not reported. If you need to debug an expression, use the **Debug Probe** where exceptions will be reported.

6.9.4. Problems Handling Values

The Wing debugger tries to handle debug data as gently as possible to avoid entering into lengthy computations or triggering errors in the debug process while it is packaging debug data for transfer. Even so, not all debug data can be shown on the display. This section describes each of the reasons why this may happen:

Wing may time out handling a value -- Large data values may hang up the debug server process during packaging. Wing tries to avoid this by carefully probing an object's size before packing it up. In some cases, this does not work and Wing will wait for the data for the duration set by the **Network Timeout** preference and then will display the variable value as `<network timeout during evaluate>`.

Wing may encounter values too large to handle -- Wing will not package and transfer large sequences, arrays or strings that exceed the size limits set by **Huge List Threshold** and **Huge String Threshold** preferences. On the debugger display, oversized sequences and arrays are annotated as `huge` and `<truncated>` is prepended to large truncated strings.

To avoid this, increase the value of the threshold preferences, but be prepared for longer data transfer times. Note that setting these values too high will cause the debugger to time out if the **Network Timeout** value isn't also increased.

An alternative available in Wing IDE Professional for viewing large data values is to enter expressions into the **Watch tool** or **Debug Probe** to view sub-parts of the data rather than tranferring the whole top-level portion of the value.

Wing may encounter errors during data handling -- Because Wing makes assignments and comparisons during packaging of debug data, and because it converts debug data into string form, it may execute special methods such as `__cmp__` and `__str__` in your code. If this code has bugs in it, the debugger may reveal those bugs at times when you would otherwise not see them.

The rare worst case scenario is crashing of the debug process if flawed C or C++ extension module code is invoked. In this case, the debug session is ended.

More common, but still rare, are cases where Wing encounters an unexpected Python exception while handling a debug data value. When this happens, Wing displays the value as `<error handling value>`.

These errors are not reported as normal program errors in the Exceptions tool. However,

extra output that may contain the exception being raised can be obtained by setting the **Debug Internals Log File** preference.

Stored Value Errors

Wing remembers errors it encounters on debug values and stores these in the project file. These values will not be refetched during subsequent debugging, even if Wing is quit and restarted.

To override this behavior for an individual value, use the `Force Reload` item in the right-click popup menu of a tree view variable area.

To clear the list of all errors previously encountered so that all values are reloaded, use the `Clear Stored Value Errors` item in the `Debug` menu. This operates only on the list of errors known for the current debug file, if a debug session is active, or for the main debug file, if any, when no debug process is running.

6.10. Interactive Debug Probe

The debug probe acts like the Python shell for evaluating and executing arbitrary Python code in the context of a debug program. This acts on the current debug stack frame, and thus is available only when the debug program is paused.

You may use many of Wing's source editor commands and key bindings within the debug probe, and can use the up/down arrow keys to traverse a history of recently typed commands.

If commands you type change any local, instance, or global data values, cause modules to be loaded or unloaded, set environment variables, or otherwise alter the run environment, your debug program will continue within that altered state. All visible variable display views are also updated after each line entered in the debug probe in order to reflect any changes caused by your commands. Since you may not notice these changes, caution is needed to avoid creating undesired side-effects in the running debug program.

One limitation is that private instance variables prefixed by double underscore (such as `self.__my_var`) cannot be directly inspected or altered within the interactive shell. Python will report that the attribute is not defined because it internally prefixes the class name where the private variable is found. These can easily be viewed using the Stack Data tool. Alternatively, use the fully qualified form of the private instance variable name: For example, to access `__my_var` in an instance of class `myclass`, use `self._myclass__my_var` in the debug probe.

Note that in this version of Wing, breakpoints are never reached as a result of entries typed into the debug probe, and any exceptions are reported only after the fact. This means that activity in the debug probe has no effect on the debug run position or stack, even though an exception location in source code may in some cases be displayed.

Preference **Raise Source from Tools** can be used to determine whether source code windows will be raised when exceptions occur in the debug probe.

The `Options` menu in the Debug Probe provides the following options:

- **Clear** -- Clear previous text from the shell.
- **Save a Copy** -- Save a copy of the shell to a disk file.
- **Wrap Lines** -- Toggle whether or not long lines are wrapped in the display.

6.11. Interactive Python Shell

A Python Shell is provided for execution of commands and evaluation of expressions outside of your debug program.

Since this shell runs a separate Python process that is independent of your debug process, it is always enabled and functions without regard to the state of any running debug process.

The Python Shell always runs the same version of Python as is used for your debug process. This is described in more detail in section **Debug Properties**

To clear the state of the Python shell, press the New Session button. This will terminate the external Python process and restart it, thus clearing and resetting the state of the shell.

Convenient ways to run parts of your source code in the shell include:

Copy/Paste part of a file -- Wing will automatically adjust leading indentation so the code can be executed in the shell.

Drag and Drop part of a file -- This works like Copy/Paste.

Evaluate File in Python Shell -- This command in the `Source` menu will evaluate the top level of the current file in the shell.

Evaluate Selection in Python Shell -- The command in the `Source` menu and editor's context menu (right-click) will evaluate the current selection in the shell.

Options menu -- This menu in the Python Shell tool contains items for evaluating the current file or selection, and some options for controlling evaluation: `Evaluate Whole Lines` causes Wing to round up the selection to the nearest line when evaluating selections, making it easier to select the desired range; `Auto-restart when Evaluate File` causes Wing to automatically restart the shell before evaluating a file, so that each evaluation is made within a clean new environment.

The `Options` menu also allows restarting the shell on demand, saving a copy to a file, and controlling whether or not Wing wraps long lines in the display.

6.12. Managing Exceptions

Wing's debugger tries to detect unhandled exceptions as they are raised in your debug process and will report them to you immediately, making it possible to view the program state that led to the exception and to step through subsequently reached `finally` clauses. This is done by looking up the stack for exception handlers written in Python, and reporting only exceptions for which there is no matching handler.

This technique works well with wxPython, PyGTK, and in most other code where unexpected exceptions either lead to program termination or are handled by catch-all exception handlers written in C/C++ extension module code.

For code with catch-all exceptions written in Python, Wing may fail to report unexpected exceptions unless those handlers are rewritten as described in **Trouble-shooting Failure to Stop on Exceptions**.

In some cases, Wing's unhandled exception detector can report normal handled exceptions that are not seen outside of the debugger. This occurs when the exceptions are handled in C/C++ extension module code. Wing can be trained to ignore these by checking the `Ignore this exception location` check box in the debugger's Exception tool. Ignored exceptions are still reported if they actually lead to program termination, and your selection is remembered in your project file so only needs to be made once. Use `Clear Ignored Exceptions` from the `Debug` menu at any time to reset the ignore list to blank.

The **Exception Reporting** preference can be used to turn off Wing's predictive exception handler in favor of other policies for deciding which runtime exceptions should be reported. The following exception reporting choices are available:

Immediately if Appear Unhandled -- This is the default described above. The debugger will stop on exceptions immediately when they are raised, but only if no handler can be found for that exception. False positives can be ignored from the `Exceptions` tool.

Always Immediately -- The debugger will stop at every single exception immediately when it is raised. In most code this will be very often, since exceptions may be used internally to handle normal, acceptible runtime conditions.

At Process Termination -- In this case, the debugger will make a best effort to stop and report exceptions that actually lead to process termination. This occurs just before or sometimes just after the process is terminated. The exception is also printed to `stderr`, as it would be when running outside of the debugger.

When working with an **Externally Launched Debug Process** , the `At Process Termination` mode may not be able to stop the debug process before it exits, and in some cases may even fail to show any post-mortem traceback at all (except as printed to `stderr` in the debug process).

Similarly, when working with wxPython, PyGTK, and similar environments that include a catch-all exception handler in C/C++ code, the `At Process Termination` mode will fail to report any unexpected exceptions occurring during the main loop because those exceptions do not actually lead to process termination.

For both of these cases, we strongly recommend using the `Immediately if Appear Unhandled` exception reporting mode.

6.13. Debug Process I/O

While running under the Wing debugger, any output from `print` or any writes to `stdout` or `stderr` will be seen in the Debug I/O tool. This is also where you enter keyboard input, if your debug program requests any with `input()` or `raw_input()` or by reading from `stdin`.

The code the services debug process I/O does two things: (1) any waits on `sys.stdin` are multiplexed with servicing of the debug network socket, so that the debug process remains responsive to Wing IDE while waiting for keyboard input, and (2) in some cases, I/O is redirected to another window.

For a debug process launched from within Wing, keyboard I/O always occurs either in the Debug I/O tool or in a new external console that is created before the debug process is started. This can be controlled as described in **External I/O Consoles**

Debug processes launched outside of Wing, using `wingdbstub`, always do their keyboard I/O through the environment from which they were launched (whether that's a console window, web server, or any other I/O environment).

When commands are typed in the **Debug Probe**, I/O is redirected temporarily to the Debug Probe only during the time that the command is being processed.

6.13.1. External I/O Consoles

By default, Wing IDE collects output from your debug process into the Debug I/O tool. This is also where you can enter keyboard input, if your debug program requests any.

In cases where the debug process requires specific characteristics provided by the Windows Console or specific Linux/Unix shell, you can redirect debug I/O to a new external window using the **Use External Console** preference.

The most effective way to keep the external console visible after the debug process exits is to place a breakpoint on the last line of your program. Alternatively, set the **External Console Waits on Exit** preference to True. However, this can result in many external consoles being displayed at once if you do not press enter inside the consoles after each debug run.

On Linux/Unix it is possible to select which console applications will be tried for the external console by altering the **External Consoles** preference.

Windows always uses the standard DOS Console that comes with your version of Windows.

6.13.2. Disabling Debug Process I/O Multiplexing

If you are using only Python-level I/O calls in your program, you will not need to be aware of the way Wing alters your debug program's I/O environment, because it mimics the environment found outside of the debugger. There are however several cases that can affect users that bypass Python-level I/O by doing C/C++ level I/O from within an extension module:

- Any C/C++ extension module code that does standard I/O calls using the C-level `stdin` or `stdout` will bypass Wing's I/O environment (which affects only Python-level `stdin` and `stdout`). This means that waiting on `stdin` in C or C++ code will make the debug process unresponsive to Wing, causing time out and termination of the debug session if you attempt to Pause or alter breakpoints at that time. In this case, redirection of I/O to the debugger I/O tool and Debug Probe will also not work.

- On all platforms, calling C-level `stdin` from multiple threads in a multi-threaded program may result in altered character read order when running under the Wing debugger.

- When debugging on win32, calling C-level `stdin`, even in a single-threaded program, can result in a race condition with Wing's I/O multiplexer that leads to

out-of-order character reads. This is an unavoidable result of limitations on multiplexing keyboard and socket I/O on this platform.

If you run into a problem with keyboard I/O, you should:

1) Turn off Wing's I/O multiplexer by setting the **Use sys.stdin Wrapper** preference to `False`.

2) Turn on the **Use External Console** preference (for details see **External I/O Consoles**)

Once that is done, I/O should work properly in the external console, but the debug process will remain unresponsive to Pause or breakpoint commands from Wing IDE whenever it is waiting for input, either at the C/C++ or Python level.

Also, in this case keyboard input invoked as a side effect of using the Debug Probe will happen through unmodified `stdin` instead of within the Debug Probe, even though command output will still appear there.

6.14. Attaching and Detaching

Debug processes normally contact Wing IDE automatically during startup. However, Wing IDE can also attach to debug processes that are already running but not yet in contact with the IDE if the process will allow it. There are two cases where this is useful:

(1) When an externally launched process (one that uses `wingdbstub.py`, as described in section **Debugging Externally Launched Code**) cannot reach the IDE at the configured host and port during initial startup, for example because the IDE is not yet running or was not configured to accept debug connections.

(2) When a process attached to the IDE is disconnected using `Detach from Process` in the Debug menu or the detach icon in the toolbar.

In either case, the IDE can manage any number of detached processes, allowing you to attach to any one process at a time.

6.14.1. Access Control

Wing will not allow attach/detach functionality unless it has available to it a password that can be used to control access. This is very important because an unsecured debug server provides the client (Wing IDE) full control of the host machine via the Debug

Probe tool. Any Python command can be executed in this way, including programs that compromise the security of your machine and network.

Because Wing sets up an access control password during installation, attach and detach will work out of the box as long as your debug processes are launched from Wing IDE, by you from the command line, or in the context of some service or program that is running under your user name on a machine that has access to your **User Settings Directory**.

If you plan to debug remotely, you will also need to copy the file `.wingdebugpw` from your **User Settings Directory** into the same directory as `wingdbstub.py`.

6.14.2. Detaching

The `Detach from Process` item in the Debug menu is used to detach from an active debug process.

Whenever a process is detached, it continues running as if outside of the debugger, without stopping at any breakpoints or exceptions. Even if a process is paused within the debugger at time of detaching from the IDE, the process will start running actively immediately after the IDE disconnects.

6.14.3. Attaching

The `Attach to Process` item in the Debug menu is available whenever no other debug process is attached to the IDE. This brings up a dialog box that includes a list of available processes to attach to. The list is built from hard-wired host/port pairs given with the **Common Attach Hosts** preference, combined with known processes that were previously attached to Wing IDE.

Wing updates the list of available processes as debug sessions are terminated from the IDE, as they are seen to exit from the outside while attached to Wing, or when the process cannot be contacted by Wing.

To attach to a process, select it from the list and push the Attach button. You may also type in a host/port value manually if your choice is not on the list (see **Identifying Foreign Processes**).

Once you are attached to a process, it continues running until it reaches a breakpoint, unhandled exception, or you `Pause` it.

6.14.4. Identifying Foreign Processes

When debugging externally launched code (as described in **Debugging Externally Launched Code**), you may use the `kAttachPort` constant in `wingdbstub.py` to set the port on which the debug process will listen for attach requests from Wing IDE. This is useful when spawning multiple processes concurrently, or in other cases where the debug process may not be able to attached to Wing IDE as it starts up.

It is important to set unique values for the `kAttachPort` value for each concurrent, externally-launched process. If the set port is in use, a random port number will be used instead and it may be difficult to determine this number if the process cannot initially contact Wing IDE to register itself.

Once this is done, the debug process can be reached from Wing IDE by typing its host/port into the Attach dialog text areas. If you find yourself typing a host/port value often, it is best to add that value to the **Attach Defaults** preference.

See section **Debugging Externally Launched Code** for more information.

6.14.5. Constraints

Wing supports attaching only to a single debug process at a time. Whenever you detach from a process, it begins free-running and will not stop at any breakpoints or non-fatal exceptions. This limits what can be done with detach/attach from a single copy of Wing. If you wish to actively debug two processes at once, simultaneously controlling stepping, breakpoint activation, and execution (as in a client/server network program), you must run two copies of Wing at once.

6.15. Debugging Externally Launched Code

This section describes how to start debugging from a process that is not launched by Wing. Examples of debug code that is launched externally include CGI scripts or web servlets running under a web server and embedded Python scripts running inside a larger application.

6.15.1. Importing the Debugger

The following step-by-step instructions can be used to start debugging in externally launched code that is running on the same machine as Wing IDE:

1) Copy `wingdbstub.py` from the Wing IDE installation directory into the same directory as your debug program.

2) In some cases, you will also need to copy the file `.wingdebugpw` from your **User Settings Directory** into the same directory as `wingdbstub.py`. This is needed when running the debug process as a different user or in a way that prevents the debug process from reading the `.wingdebugpw` file from within your profiles directory.

3) At the point where you want debugging to begin, insert the following source code: `import wingdbstub` Depending on your code base, you may need to be cautious about whether this statement is reached by multiple processes or threads. If this happens, the first instance will connect to Wing and the second one will fail to connect and continue running without debug.

4) Make sure the Wing IDE preference **Enable Passive Listen** is set to `True` to allow connection from external processes.

5) Set any required breakpoints in your Python source code.

6) Initiate the debug program from outside Wing IDE, for example with a page load in your web browser, if the program is a CGI script. You should see the status indicator in the Stack Data, Watch, and Debug Probe tools change from red to yellow or green as described in **Debugger Status**. Make sure that you are running the Python interpreter without the `-O` option. The debugger will not work when optimization is turned on.

7) The debugger should stop at the first breakpoint or exception found. If no breakpoint or exception is reached, the program will run to completion, or you can use the Pause command in the Debug menu.

Enabling Process Termination
In some cases, you may wish to enable termination of debug processes that were launched from outside of Wing IDE. By default, Wing recognizes externally launched processes and disables process termination in these cases unless the **Kill Externally Launched** preference is set to `True`.

If you have problems making this work, try setting `kLogFile` variable in `wingdbstub.py` for log additional diagnostic information.

6.15.2. Debug Server Configuration

In some cases you may also need to alter other preset configuration values at the start of `wingdbstub.py`. These values completely replace any values set in Wing's Project

or File Properties, which are relevant only when the debug program is launched from within Wing. The following options are available:

- The debugger can be disabled entirely with `kWingDebugDisabled=1`. This is equivalent to setting the `WINGDB_DISABLED` environment variable before launching the debug program.

- Set `kWingHostPort` to specify the network location of Wing IDE, so the debugger can connect to it when it starts. This is equivalent to setting the `WINGDB_HOSTPORT` environment variable before launching the debug program. The default value is `localhost:50005`. See section **Remote Debugging** for details if you need to change this value.

- You can control whether or not the debugger's internal error messages are written to a log file by setting `kLogFile`. Use `<stdout>`, `<stderr>`, or a file name. If the given file doesn't exist, it is created if possible. Note that using `<stderr>` may cause problems on Windows if the debug process is not running in a console. This is equivalent to setting the `WINGDB_LOGFILE` environment variable before launching the debug program (use a value of `-` to turn off logging to file).

- Set `kEmbedded` to `1` when debugging embedded scripts. In this case, the debug connection will be maintained across script invocations instead of closing the debug connection when the script finishes. When this is set to `1`, you must call `wingdbstub.debugger.ProgramQuit()` before your program exits in order to cleanly close the debug connection to the IDE. This is equivalent to setting the environment variable `WINGDB_EMBEDDED`.

- Set `kAttachPort` to define the default port at which the debug process will listen for requests to attach (not available in Wing IDE Personal). This is equivalent to setting the `WINGDB_ATTACHPORT` environment variable before launching the debug program. If this value is less than 0, the debug process will never listen for attach requests. If it is greater than or equal to 0, this value is used when the debug process is running without being in contact with Wing IDE, as might happen if it initially fails to connect to the above-defined host and port, or if the IDE detaches from the process for a period of time. For Wing IDE Professional, this is described in more detail in section **Attaching and Detaching**.

- Set `kPWFilePath` and `kPWFileName` to define the search path and file name used to find a `.wingdebugpw` file for the debugger. The environment variables `WINGDB_PWFILEPATH` and `WINGDB_PWFILENAME` will override these settings. The file path should be a Python list of strings if set in `wingdbstub.py` or a list of directories separated by the path separator (`os.pathsep`) when sent by environment variable. The string `$<winguserprofile>` may be used to specify Wing's

User Settings Directory for the user that the debug process is running as. The password file name is usually `.wingdebugpw` but may be changed in cases where this naming is inconvenient.

- Optionally, set `WINGHOME`, which is the location of the Wing IDE distribution's home directory. This is set up during installation, but may need to be altered if you are running Wing from source or copied the debugger binaries over from another machine.

Setting any of the above-described environment variable equivalents will override the value given in the `wingdbstub.py` file.

Behavior on Failure to Attach to IDE
Whenever the debugger cannot contact Wing IDE (for example, if the IDE is not running or is listening on a different port), the debug program will be run without debugging. This is useful since debug-enabled CGIs and other programs should work normally when Wing is not present. However, you can force the debug process to exit in this case by setting the `kExitOnFailure` flag in `wingdbstub.py`. To attach to processes started without debug, see **Attaching** (only available in Wing IDE Professional).

6.15.3. Remote Debugging

Since remote debugging is fairly complicated to configure, we currently recommend using remote display of the IDE via X Windows (Linux/Unix) or Remote Desktop (Windows) instead of setting up the IDE on a separate host from the debug process.

When this is not an option, you can also ask the debugger to connect remotely over the network. In order to do this, take the following steps (see also **Remote Debugging Example**):

(1) First set up Wing IDE to successfully accept connections from another process within the same machine, as described in section **Importing the Debugger**. You can use any Python script for testing this until you have values that work.

(2) Optionally, alter the **Server Host** preference to the name or IP address of the network interface on which the IDE listens for debug connections. The default server is `None`, which indicates that the IDE should listen on all the valid network interfaces on the host.

(3) Optionally, alter the preference **Server Port** to the TCP/IP port on which the IDE should listen for debug connections. This value may need to be changed if multiple copies of Wing IDE are running on the same host.

(4) Set the **Allowed Hosts** preference to include the host on which the debug process will be run. For security purposes, Wing will reject connections if the host isn't included here.

(5) Configure any firewall on the system that Wing IDE is running on to accept a connection on the server port from the system that the debug process will run on.

(6) Next install Wing IDE on the machine on which you plan to run your debug program. Creating an entire Wing IDE installation is the easiest approach.

An alternative is to copy only the debug server code out of your primary Wing installation. This encompasses all of the following files and directories under `WINGHOME`:

~~

bin/wingdb.py bin/#.#/src/debug/server bin/#.#/src.zip/debug/server (only Python >= 2.3) bin/#.#/opensource/schannel (only Python < 2.3) bin/#.#/opensource.zip/schannel (only Python >= 2.3)

Replace #.# with 1.5, 2.0, 2.1, 2.2, 2.3, and 2.4 (one for each supported version of Python). If you're only using one version of Python, you can omit the directories for the versions that you are not using.

The directories within zip files (used only in Python 2.3 and later) can either be copied by moving the entire zip file or by creating a subset that contains only the necessary directories.

Be sure to copy these directories from a Wing installation on the same type of host, so that on Linux/Unix you include `*.so` extension modules, on Windows `*.pyd` extension modules, and so forth.

(7) Next, transfer copies of all your debug code so that the source files are available on the host where Wing IDE will be running and at least the `*.pyc` files are available on the debug host.

During debugging, the client and server copies of your source files must match or the debugger will either fail to stop at breakpoints or stop at the wrong place, and stepping through code may not work properly.

Since there is no mechanism in Wing IDE for transferring your code, you need to use NFS, Samba, FTP or some other file transfer mechanism to keep the remote files up to date as you edit them in Wing.

If files appear in different disk locations on the two machines, you will also need to set up a file location map, as described in **File Location Maps**.

(8) On your debug host, copy `wingdbstub.py` into the same directory as your source files

and import it in your Python source as described in **Debugging Externally Launched Code**.

(9) If you didn't copy `wingdbstub.py` out of a complete installation of Wing IDE on the debug host, you will need to set `kWingHome` to match the location where you have copied the debug server code on your debug host.

(10) In `wingdbstub.py` on your debug host, set `kWingHostPort`. The host in this value must be the IP address of the machine where Wing IDE is running. The port must match the port configured with the **Server Port** preference on the host where Wing IDE is running.

(11) Then restart Wing and try running your program on the debug host. You should see the Wing IDE debugger status change to indicate that a debug process has attached.

If you have problems making this work, try setting `kLogFile` variable in `wingdbstub.py` for log additional diagnostic information.

6.15.4. File Location Maps

In cases where the full path to your source is not the same on both machines, you also need to set up a mapping that tells Wing where it can find your source files on each machine.

This is done with the **Location Map** preference, which lists corresponding local and remote directory locations for each remote host's dotted quad IP address.

One of the host IP addresses within this preference can be set to `"*"` to define a default mapping for all hosts that are not otherwise specified in the location map.

Each host IP address in the location map is paired with one or more `(remote_prefix, local_prefix)` tuples. The remote file prefix will be a full path on the debug server's file system. The local file prefix should be a URL, optionally starting with `file:` (this URL should not contain backslashes (\), even if the local host is a Windows machine) or a UNC `\\server\share\dir` style path name.

The best way to understand this is to look at the **Location Map Examples**.

When running Wing IDE on Windows XP, UNC formatted file names such as `\\machine\path\to\file` may be used. On other Windows systems, you must map remote drives to a drive letter such as `F:`. In cases where setting up a persistent drive mapping is a problem, use a `cmd.exe` script with a `net use` command to map the drive on demand.

Note that making symbolic links on the client or server will not work as an alternative to using this mapping. This is a side-effect of functionality in the debugger that ensures

that debugging works right when symbolic links are present: Internally, source file names are always resolved to their actual full path location.

6.15.4.1. File Location Map Examples

The best way to understand location mapping is to inspect a few examples.

Defaults Explained

The default value for the **Location Map** preference contains one entry for `127.0.0.1` where the mapping is set to `None` (in Python this is represented as `{'127.0.0.1':None}`). This is equivalent to the more verbose Python representation of `{'127.0.0.1':[('/','file:')]}`. It converts full paths on the debug server to the client-side URLs without altering any part of the full path.

Two Linux/Unix Hosts

Here is an example setting for `debug.location-map` that would be used if running Wing on `desktop1` and debugging some code on `server1` with IP address `192.168.1.1`:

```
debug.location-map={
  '127.0.0.1':None,
  '192.168.1.1':[('/home/apache/cgi', 'file:/svr1/home/apache/cgi')]
}
```

In this example, the files located in `/home/apache/cgi` on `server1` are the same files seen in `/server1/home/apache/cgi` on `desktop1` because the entire file system on `server1` is being shared via NFS and mounted on `desktop1` under `/svr1`.

To enter this value in Preferences, you would add `192.168.1.1` as a new Remote IP Address and a single local/remote mapping pair containing `/home/apache/cgi` and `file:/svr1/home/apache/cgi`.

IDE on Linux/Unix with Debug Process on Windows

If you are debugging between Windows and Linux or Unix, some care is needed in specifying the conversion paths because of the different path name conventions on each platform. The following entry would be used when running Wing IDE on a Linux/Unix host and the debug process on a Windows host with ip address `192.168.1.1`:

```
debug.location-map={
  '127.0.0.1':None,
  '192.168.1.1':[(r'e:\src', 'file:/home/myuser/src')],
}
```

In this example the Linux/Unix directory `/home/myuser` is being shared via Samba to the Windows machine and mapped to the `e:` drive.

In the Preferences GUI, you would add `192.168.1.1` as a new Remote IP Address and a single local/remote mapping pair containing `e:\src` and `file:/home/myuser/src`.

IDE on Windows with Debug Process on Linux/Unix

If running Wing IDE on a Windows host and the debug process on a Linux/Unix host with IP address `192.168.1.1`, the following would be used instead for the same file locations:

```
debug.location-map={
  '127.0.0.1':None,
  '192.168.1.1':[('/home/myuser/src', 'file:e:/src')],
}
```

Again, note the use of forward slashes in the URL even though the file is on a Windows machine.

In the Preferences GUI, you would add `192.168.1.1` as a new Remote IP Address and a single local/remote mapping pair containing `/home/myuser/src` and `file:/e:/src`.

Two Windows Hosts

If running Wing IDE on Windows and the debug process on another Windows machine with IP address `192.168.1.1`, the following would be used:

```
debug.location-map={
  '127.0.0.1':None,
  '192.168.1.1':[(r'c:\src', 'file:e:/src')],
}
```

In this case, the host where Wing is running has mapped the entire remote (debug process) host's `c:` drive to `e:`.

In the Preferences GUI, you would add `192.168.1.1` as a new Remote IP Address and a single local/remote mapping pair containing `c:\src` and `file:e:/src`.

Two Windows Hosts using UNC Share

A UNC style path name can be used on Windows XP as follows:

```
debug.location-map={
  '127.0.0.1':None,
```

```
  '192.168.1.1':[(r'c:\src', '\\server\share\dir')],
}
```

In this case, c:src on the remote host, where the debug process is running, can be accessed as \serversharedir on the machine where Wing IDE is running.

In the Preferences GUI, you would add `192.168.1.1` as a new Remote IP Address and a single local/remote mapping pair containing `c:\src` and `\\server\share\dir`.

6.15.5. Remote Debugging Example

Here is a simple example that enables debugging a process running on a Linux/Unix host (`192.168.1.200`) using Wing IDE running on a Windows machine (`192.168.1.210`). This example is for wingdbstub users only. If you are using the WingDBG product to debug Zope code, please refer to the **Zope Debugging How-To** (also included in the WingDBG product's Help tab).

On the Windows machine, the following preferences must be specified:

- **Enable Passive Listen** should be checked
- **Server Host** should be set to `All Interfaces` (this is the default)
- **Server Port** should be set to 50005 (this is the default)
- **Allowed Hosts** should be altered by adding `192.168.1.200`

On the Linux/Unix machine, the following value is needed in `wingdbstub.py`:

```
kWingHostPort='192.168.1.210:50005'
```

Once this is done and Wing has been restarted, you should be able to run code that imports `wingdbstub` on the Linux/Unix machine and see the debug connection establish on the Windows machine.

Then you will need to set up file sharing between the two machines (for example, using Samba) and will need to establish a location map in your Wing IDE preferences on the Windows machine.

If your source code on the Linux/Unix machine is in `/home/myuser/mysource` and you map `/home/myuser` to `e:` on the Windows machine, then you would use the following location map in conjunction with the above settings:

```
debug.location-
map=('192.168.1.200': [('/home/myuser/mysource', \
                                          'file:e:/mysource')]}
```

To enter this location map via the Preferences GUI, you would add `192.168.1.200` as a new Remote Host IP and enter a single mapping pair with `/home/myuser/mysource` and `file:e:/mysource`.

See **Location Map Examples** for additional examples.

6.15.6. Debugger API

A simple API can be used to control debugging more closely, once you have imported `wingdbstub.py` the first time, as was described in section **Importing the Debugger**.

This is useful in cases where you want to be able to start and stop debugging on the fly several times during a debug run, for example to avoid debug overhead except within a small sub-section of your code. It can also be useful in embedded scripting environments.

To use the API, take the following steps: (1) Configure and import `wingdbstub.py` as described in section **Importing the Debugger**. (2) Subsequently, use the instance variable `wingdbstub.debugger` to make any of the following calls:

- **StartDebug(stophere=0, autoquit=1, connect=1)** -- Start debugging, optionally connecting back to the IDE and/or stopping immediately afterwards. Set `autoquit=0` to avoid automatically terminating debug when program exit is detected (this is the same as setting `kEmbedded` in `wingdbstub.py`).
- **StopDebug()** - Stop debugging completely and disconnect from Wing IDE. The debug program continues executing in non-debug mode and must be restarted to resume debugging.
- **SuspendDebug()** - This will leave the connection to the debug client intact but disables the debugger so that connection overhead is avoided during subsequent execution.
- **ResumeDebug()** - This will resume debugging using an existing connection to Wing.
- **Break()** -- This pauses the free-running debug program on the current line, as if at a breakpoint.
- **ProgramQuit()** - This must be called before the debug program is exited if `kEmbedded` was set to `1` in `wingdbstub.py` or if `autoquit=0` in the preceding

`StartDebug()` API call (if any). This makes sure the debug connection to the IDE is closed cleanly.

Here is a simple usage example:

```
import wingdbstub
a = 1 # This line is debugged
wingdbstub.debugger.SuspendDebug()
x = 1 # This is executed without debugging
wingdbstub.debugger.ResumeDebug()
y = 2 # This line is debugged again
```

`SuspendDebug()` and `ResumeDebug()` can be called as many times as desired, and nested calls will be handled so that debugging is only resumed when the number of `ResumeDebug()` calls matches the number of `SuspendDebug()` calls.

6.16. Running Without Debug

Files may also be executed outside of the debugger. This can be done with any Python code, Makefiles, and any other file that is marked as executable on disk. This is done with the `Execute Current File` and `Execute Recent` items in the `Debug` menu, or with `Execute Selected` after right-clicking on the project view.

Files executed in this way are run in a separate process and any input or output occurs within the window from which Wing was launched (or is entirely hidden if Wing was launched from a desktop icon).

This is useful for triggering builds, executing utilities used in development, or even to launch a program that is normally launched outside of Wing and debugged using `wingdbstub.py`.

Note that files executed in this way are always invoked as if from their enclosing directory and without any parameters. There is currently no facility for specifying parameters or redirecting input/output.

6.17. Debugger Limitations

There are certain situations that the debugger cannot handle, because of the way the Python programming language works. If you are having problems getting the debugger

to stop at breakpoints or to display source as you step through your code, one or more of these may apply.

Always read the **Trouble-shooting Failure to Debug** section first. If that fails to uncover your problem, refer to the following detailed documention of debugger limitations:

(1) Your source files must be stored on disk and accessible to the IDE. If you are trying to debug code fragments, try writing them to disk temporarily and setting the `__file__` variable in the module name space before invoking Python's `exec` or `eval`.

(2) Running without saving will lead to incorrect display of breakpoints and run position because the debug process runs against the on-disk version of the source file. Wing will indicate in the Messages tool and Stack Data status indicator that some files are out of `sync` so this case should only occur if you ignore its warnings.

(3) You cannot run the debug program using the `-O` or `-OO` optimization options for the Python interpreter. This removes information about line numbers and source file names, making it impossible to stop at breakpoints or step through code.

(4) There are several cases where Wing may fail to stop at breakpoints or exceptions, or may fail to find source files corresponding with breakpoints or exception points. All of these are caused by storage of incorrect file names in `*.pyc` files:

- Moving `*.pyc` files on disk after they are generated invalidates the file name stored in the file if it is a partial relative path. This happens if your `PYTHONPATH` or `sys.path` contains partial relative path names.
- A similar problem may result from use of `compileall.py` and some other utilities that don't record a correct filename in the `*.pyc` file.
- If you run the same code twice using different paths to the same working directory, as is possible on Linux/Unix with symbolic links, the file names left in `*.pyc` may contain a mix of each of these paths. If the symbolic link that was used is subsequently removed, some of the file names become invalid.

The fix for all of these problems is to remove the `*.pyc` files and let Python regenerate them from the corresponding `*.py` files with the correct file name information.

Hint: You can open `*.pyc` files in most text editors to inspect the stored file names.

(5) For code that spends much of its time in C/C++ without calling Python at all, for example as in a GUI main loop, the debugger may not reliably stop at breakpoints added during a run session, and may not respond to Pause requests. See section **Debugging Non-Python Mainloops** for more information.

(6) You cannot use `pdb` in code that you are running within the Wing debugger. The two

debuggers conflict because they attempt to use the same debugger hooks in the Python interpreter.

(7) If you override `__import__` in your code, you will break the debugger's ability to stop at breakpoints unless you call the original `__import__` as part of your code whenever a module is actually imported. If you cannot call the original `__import__` for some reason, it may be possible to instead use `wingdbstub` and then call `wingdbstub.debugger.NotifyImport(mod)` from your import handler (where `mod` is the module that was just imported).

(8) If you set `__file__` in a module's name space to a value other than its original, Wing will be unable to stop at breakpoints in the module and may fail to report exceptions to the IDE's user interface.

(9) If you use an extension module to call C/C++ level `stdio` calls instead of using the Python-level facilities, the debug process will remain unresponsive to Wing IDE while waiting for keyboard input, I/O redirection to the Debug Probe will fail, and you may run into out-of-order character reads in some cases. Details can be found in **Debug Process I/O**.

(10) Using partial path names in module `__file__` attribute can in rare cases cause Wing to fail to stop on breakpoints and exceptions, to fail to display source files, or to confuse source files of the same name.

A partial path name may end up in `__file__` only when (a) invoking Python code with a partial path name, for example with `python myfile.py` instead of `python /path/to/myfile.py`, (b) sending partial path names into `exec`, (c) using partial path names in your `PYTHONPATH` or `sys.path`, or (d) using `compileall.py` or similar tool to compile modules with a partial path name.

Because Wing does everything possible to avoid this problem in practice, it actually only occurs in the following rare cases:

- When modules are loaded with partial path names and `os.chdir()` is called before debugging is started. This is only possible when using `wingdbstub` or otherwise starting debug after your debug process is started.
- When modules are loaded with partial path names and `os.chdir()` is called after `wingdbstub.debugger.SuspendDebug()` and before `wingdbstub.debugger.ResumeDebug()`.
- When modules are loaded with partial path names and removed from `sys.modules` before the debugger is started or while debugging is suspended.
- When code objects are created on the fly using `compile()`, the C API, or the new module, a relative filename or an incorrect filename are used

for the filename argument, and `os.chdir()` is called before the code is executed.

(11) Wing tries to identify when source code in the IDE matches or does not match the code that is running in the debug process. There are certain very rare cases where this will fail, which may lead to failure to stop on breakpoints and other problems even when files are identified by the IDE as being synchronized:

Using `execfile()`, `eval()`, or `exec` with a globals dict that contains `__file__` will cause Wing to incorrectly assert that the specified file has been reloaded. In practice, this scenario usually occurs when `execfile()` is called from the top level of a module, in which case the module is in fact being loaded or reloaded (so no mis-identification of module load status occurs). However, in cases where a module load takes a long time or involves a long-running loop at the top level, the `execfile()`, `eval()`, or `exec` may occur **after** edits to the module have been made and saved. In this case, Wing will mis-identify the module as having been reloaded with the new edits.

This problem can also be triggered if a `globals` with `__file__` is explicitly passed to `execfile()`, `eval()`, or `exec`. However, it will only occur in this case when the code object file name is `?`, and `locals` and `globals` dictionaries are the same, as they are by default for these calls.

(12) In very rare cases, when using the `wingdbstub.py`, if you set `sys.exitfunc` after debugging has been started, the IDE will time out on a broken network connection after the debug program exits on an exception. This only happens for exceptions that look like they will be handled because a try/except block is present that might handle the exception, but where the exception is not in the end handled and the debug program exits without calling `StopDebug()`. Work-arounds include setting `sys.exitfunc` before importing `wingdbstub.py` or adding a top-level try/except clause that always calls `StopDebug()` before exiting the debug program.

(13) Naming a file `<string>` will prevent the debugger from debugging that file because it is confused with the default file name used in Python for code that is not located in a file.

(14) The debugger may fail to step or start after stopping at a breakpoint if the floating point mode is set to single precision (24 bit) on Intel x86 and potentially other processors. This is sometimes done by graphics libraries such as DirectX or by other code that optimizes floating point calculalations.

Scripting and Extending Wing IDE

Wing IDE provides an API that can be used to extend and enhance the IDE's functionality with scripts written in Python.

Simple scripts can be written without any extra tools -- Wing will find and load scripts at startup and reload them if they change on disk. The API Wing provides allows scripts access to the editor, debugger, project, and a range of application-level functionality. Scripts may also access all **documented preferences** and can issue any number of **documented commands** which implement functionality not duplicated in the formal Python API.

Scripts can be executed like any other command provided by Wing IDE. Scripts can add themselves to the editor and project context menus, or to new menus in the menu bar, and they can also register code for periodic execution as an idle event. They can also be bound to a key combination, or can be invoked by name using the `command-by-name` command.

Errors encountered while loading or executing scripts are displayed in the Scripts channel of the **Messages** tool.

More advanced scripting, including the ability to add tool panels, is also available but generally requires running a copy of Wing IDE from source code, so that scripts can be debugged more efficiently.

7.1. Scripting Example

The scripting facility is documented in detail in the sections that follow, but in most cases it is easiest simply to work from the examples in the `scripts` directory in the Wing IDE installation, using the rest of this chapter as a reference.

User scripts are usually placed inside a directory named `scripts` within the **User Settings Directory**. They can also be placed in `scripts` inside the Wing IDE installation.

Try adding a very simple script now by pasting the following into a file called `test.py` within one of the `scripts` directories:

```
import wingapi
def test_script(test_str):
  app = wingapi.gApplication
  v = "Product info is: " + str(app.GetProductInfo())
  v += "\nAnd you typed: %s" % test_str
  wingapi.gApplication.ShowMessageDialog("Test Message", v)
```

Then select Reload All Scripts from the Edit menu. This is only needed the first time a new script file is added, in order to get Wing to discover it. Afterward, Wing automatically reloads scripts whenever they are saved to disk.

Next execute the script by typing `Ctrl-F12` and then `test-script` + `Enter` into the text entry that appears in the status area. Wing will ask for the argument `test_str` using it's builtin argument collection facility. Type a string and then `Enter`. The script will pop up a modal message dialog.

Next make a trivial edit to the script (e.g., change "And you typed" to "Then you typed"). Save the script and execute the script again. You will see that Wing has automatically reloaded the script and the new text appears in the message dialog.

Finally, make an edit to the script that introduces an error into it. For example, change `import wingapi` to `import wingapi2`. Save the script and Wing will show a clickable traceback in the `Scripts` channel of the **Messages** tool. This makes it easy to quickly find and fixed errors in scripts during their development.

That's all there is to basic scripting. The most relevant examples for most simple scripts can be found in `editor-extensions.py` in the `scripts` directory inside the Wing IDE installation. This shows how to access and alter text in the current editor, among other things.

For more advanced scripting, where a more complete debugging support is needed, you will need to obtain a copy of the Wing IDE source code distribution and run Wing from source code so that the scripts (and all of Wing) can be debugged with another copy of Wing (usually your binary installation of Wing).

7.2. Getting Started

Scripts are Python modules or packages containing one or more Python functions. When Wing starts up, it will search all directories in the configured **Script Search Path** for

modules (`*.py` files) and packages (directories with an `__init__.py` file and any number of other `*.py` files or sub-packages).

Wing will load scripts defined in each file and add them to the command set that is defined internally. The script directories are traversed in the order they are given in the preference and files are loaded in alphabetical order. When multiple scripts with the same name are found, the script that is loaded last overrides any loaded earlier under that name.

Naming Scripts

Scripts can be referred to either by their short name or their fully qualified name (FQN).

The short name of a script is the same as the function name but with underscores optionally replaced by dashes (`cmdname.replace('_', '-')`).

The FQN of a script always starts with `.user.`, followed by the module name, followed by the short name.

For example, if a script named `xpext_doit` is defined inside a module named `xpext.py`, then the short name will be `xpext-doit` and the FQN will be `.user.xpext.xpext-doit`.

Reloading Scripts

Once script files have been loaded, Wing watches the files for changes on disk and automatically reloads them as needed. As a result, there is usually no need to restart Wing when working on a script, except when a new script file is added. In that case, Wing will not load the new script until the `reload-scripts` command (`Reload All Scripts` in the `Edit` menu) is issued or the IDE is restarted.

For details on how reloading works, see **Advanced Scripting**.

Overriding Internal Commands

Wing will not allow a script to override a command that Wing defines internally (those documented in the **Command Reference**). If a script is named the same as a command in Wing, it can only be invoked using its fully qualified name. This is a safeguard against completely breaking the IDE by adding a script.

One implication of this behavior is that a script may be broken if a future version of Wing ever adds a command with the same name. This can generally be avoided by using

appropriately descriptive and unique names and/or by referencing the command from key bindings and menus using only its fully qualified name.

7.3. Script Syntax

Scripts are syntactically valid Python with certain extra annotations and structure that are used by Wing IDE to determine which scripts to load and how to execute them.

Only functions defined at the top level of the Python script are treated as commands, and only those that start with a letter of the alphabet. This allows the use of `_` prefixed names to define utilities that are not themselves commands, and allows use of Python classes defined at the top level of script files in the implementation of script functionality.

Script Attributes

In most cases additional information about each script `def` is provided via function attributes that define the type of arguments the script expects, whether or not the command is available at any given time, the display name and documentation for the command, and the contexts in which the script should be made available in the GUI.

The following are supported:

- **arginfo** -- This defines the argument types for any arguments passed to the script. It is a dictionary from the argument name to an `ArgInfo` specification (described in more detail below) or a callable object that returns this dictionary. Argument information is used by Wing to drive automatic collection of argument values from the user. When this is missing, all arguments are treated as strings.
- **available** -- This defines whether or not the script is available. If missing, the command is always available. If set to a constant, the truth value of that constant defines availability of the script. If set to a callable object, it is invoked with the same arguments as the script itself and the return value determines availability.
- **label** -- The label to use when referring to the command in menus and elsewhere. When omitted, the label is derived from the command name by replacing underscores with a space and capitalizing each word (`cmdname.replace('_', ' ').title()`)
- **doc** -- The documentation for the script. Usually, a docstring in the function definition is used instead.

- **contexts** -- The contexts in which the script will be added in the GUI, a described in more detail below.

ArgInfo

Argument information is specified using the `CArgInfo` class in the Wing API (`wingapi.py` inside `bin` in the Wing IDE installation, although the class is imported from Wing IDE's internals) and the `datatype` and `formbuilder` modules in Wing's `wingutils` package. The source code for this class and support modules is only available in the source distribution, although most use cases are covered by the following.

`CArgInfo`'s contructor takes the following arguments:

- **doc** -- The documentation string for the argument
- **type** -- The data type, using one of the classes descended from wingutils.datatype.CTypeDef (see below for the most commonly used ones)
- **formlet** -- The GUI formlet to use to collect the argument from the user when needed. This is one of the classes descended wingutils.formbuilder.CDataGui (see below for the most commonly used ones).
- **label** -- The label to use for the argument when collected from the user. This argument may be omitted, in which case Wing builds the label as for the `label` function attribute described above.

Commonly Used Types

The following classes in `wingutils.datatype.py` cover most cases needed for scripting:

- **CBoolean** -- A boolean value. Constructor takes no arguments.
- **CType** -- A value of type matching one of the parameters sent to the constructor. For example, `CType("")` for a string, `CType(1)` for an integer, and `CType(1.0, 1)` for a float, or an integer.
- **CValue** -- One of the values passed to the constructor. For example `CValue("one", "two", "three")` to allow a value to be either `"one"`, `"two"`, or `"three"`.

- **CRange** -- A value between the first and second argument passed to the constructor. For example, `CRange(1.0, 10.0)` for a value between `1.0` and `10.0`, inclusive.

Additional types are defined in `wingutils.datatype.py`, but these are not usually needed in describing scripting arguments.

Commonly Used Formlets

The following classes in `guiutils.formbuilder.py` cover most of the data collection formlets needed for scripting:

CSmallTextGui -- A short text string entry area with optional history, autocompletion, and other options. The constructor takes the following keyword arguments, all of which are optional:

```
maxlen            -- Maximum allowed text length (-
1=any, default=80)
history           --
List of strings for history (most recent 1st) or
                     a callable that will return the his-
tory (default=None)
choices           --
List of strings with all choices, or a callable
                     that will take a fragment and re-
turn all possible
                     matches (default=None)
partial_complete  --
True to only complete as far as unique match when
                     the tab key is pressed.  Default=True.
stopchars         --
List of chars to always stop partial completion.
                     Default=''
allow_only        --
List of chars allowed for input (all others are
                     not processed).  Set to None to al-
low all.  Default=None
auto_select_choice --
True to automatically select all of the entry text
                     when browsing on the autocom-
pleter (so it gets erased
                     when any typing happens).  Default=False.
```

```
default           -- The default value to use.  Default=''
select_on_focus   --
True to select range on focus click; false to retain
                     pre-focus selection.  Default=False
editable          -- True to allow editing this field.  De-
fault=True.
```

CLargeTextGui -- A longer text string. The constructor takes no arguments.

CBooleanGui -- A single checkbox for collecting a boolean value. The constructor takes no arguments.

CFileSelectorGui -- A keyboard-driven file selector with auto-completion, optional history, and option to browse using a standard file open dialog. The constructor takes the following keyword arguments:

```
want_dir          --
True to browse for a directory name (instead of a
                     file name).  Default=False.
history           --
Optional list with history of recent choices, most
                     recent first.  Default=()
default           -- The default value to use.  Default=''
```

Additional formlet types are defined in `guiutils.formbuilder.py` but these are not usually needed in collecting scripting arguments.

CPopupChoiceGui -- A popup menu to select from a range of values. The constructor takes a list of items for the popup. Each item may be one of:

```
None                 -- A divider
string               --
The value.  The label used in the menu is derived:
                        label = value.replace('_', ' ').title()
(value, label)       -- The value and label to use in menu.
(value, label, tip)  --
The value, label, and a tooltip to show when the
                        user hovers over the menu item.
```

CNumberGui -- A small entry area for collecting a number. The constructor takes these arguments (all are required):

```
min_value            -- The minimum value (inclusive)
```

```
max_value            -- The maximum value (inclusive)
page_size            --
Increment when scroller is used to browse the range
num_decimals         -- Number of decimal places (0 to col-
lect an integer)
```

Additional formlets for collecting data are defined in `guiutils.formbuilder.py`, but these are not usually needed for scripting.

Magic Default Argument Values

Wing treats certain defaults values specially when they are specified for a script's arguments. When these default values are given, Wing will replace them with instances of objects defined in the API. This is a convenient way for the script to access the application, debugger, current project, current editor, and other objects in the API. All the default values are defined in the `wingapi.py` file, as are the classes they reference.

- **kArgApplication** -- The `CAPIApplication` instance (this is a singleton).
- **kArgDebugger** -- The currently active `CAPIDebugger`.
- **kArgProject** -- The currently active `CAPIProject`.
- **kArgEditor** -- The currently active `CAPIEditor`.
- **kArgDocument** -- The `CAPIDocument` for the currently active editor.

GUI Contexts

Scripts can use the `contexts` function attribute to cause Wing to automatically place the script into certain menus or other parts of the GUI. The following contexts are currently supported (they are defined in `wingapi.py`):

- **kContextEditor** -- Adds an item to the end of the editor's context menu (accessed by right clicking on the editor)
- **kContextProject** -- Adds an item to the end of the project's context menu (accessed by right clicking on the project)
- **kContextNewMenu** -- Adds an item to a new menu in the menu bar. This is a class whose constructor takes the localized name of the menu to add. The menu is only added if one or more valid scripts with that menu context are successfully loaded.

- **kContextScriptsMenu** -- Adds an item to the scripts menu, which is shown in the menu bar if any scripts are added to it (this is currently the same as `kContextNewMenu("Scripts")` but may be moved in the future).

All scripts, under both short and fully qualified name, are always listed along with all internally defined commands in the auto-completion list presented by the `command-by-name` command (bound to keyboard sequence `Esc X` -- `Escape` key followed by `X`) and in the **Custom Key Bindings** preference.

Top-level Attributes

Default values for some of the Script Attributes defined above can be set at the top level of the script file, and some additional attributes are also supported:

- **_arginfo** -- The default argument information to use when no per-script `arginfo` attribute is present.
- **_available** -- The default availability of scripts when no `available` attribute is present.
- **_contexts** -- The default contexts in which to add scripts when no `contexts` attribute is present.
- **_ignore_scripts** -- When set to True, Wing will completely ignore this script file.
- **_i18n_module** -- The name of the `gettext` internationalized string database to use when translating docstrings in this script. See below for more information.

Importing Other Modules

Scripts can import other modules from the standard library, `wingapi` (the API), and even from Wing's internals. However, because of the way in which Wing loads scripts, users should avoid importing one script file into another. If this is done, the module loaded at the `import` will not be the same as the one loaded into the scripting manager. This happens because the scripting manager uniquifies the module name by prepending `internal_script_` so two entries in `sys.modules` will result. In practice, this is not always a problem except if global data at the top level of the script module is used as a way to share data between the two script modules. Be sure to completely understand Python's module loading facility before importing one script into another.

Internationalization and Localization

String literals and docstrings defined in script files can be flagged for translation using the `gettext` system. To do this, the following code should be added before any string literals are used:

```
import gettext
_ = gettext.translation('scripts_example', fallback=1).gettext
_i18n_module = 'scripts_example'
```

The string `'scripts_example'` should be replaced with the name of the `.mo` translation file that will be added to the `resources/locale` localization directories inside the Wing installation.

Subsequently, all translatable strings are passed to the `_()` function as in this code example:

```
kMenuName = _("Test Base")
```

The separate `_i18n_module` attribute is needed to tell Wing how to translate docstrings (which cannot be passed to `_()`).

Currently, the only support provided by Wing for producing the `*.po` and `*.mo` files used in the `gettext` translation system is in the build system that comes with the Wing IDE sources. Please refer to `build-files/wingide.py` and `build-files/README.txt` for details on extracting strings, merging string updates, and compiling the `*.mo` files. On Linux, KDE's `kbabel` is a good tool for managing the translations.

7.4. Scripting API

Wing's formal scripting API consists of several parts:

1) The contents of the `wingapi.py` file in `bin` inside the Wing IDE installation (this file is located in `src` when working from the source distribution). Please refer to the file itself for details of the API.

2) The portions of the `wingutils.datatype` and `guiutils.formbuilder` modules that are documented in the preceding section.

3) All of the **documented commands** which can be invoked using the `ExecuteCommand()` method on `wingapi.gApplication`. Note keyword

arguments can be passed to commands that take them, for example `ExecuteCommand('replace-string', search_string="tset", replace_string="test")`

4) All of the **documented preferences** which can be obtained and altered using `GetPreference` and `SetPreference` on `wingapi.gApplication`.

Scripts can, of course, also import and use standard library modules from Python, although Wing ships with a pruned subset of the standard library that includes only those modules that are used by the IDE's internals.

Advanced scripts may also "reach through" the API into Wing internals, however this requires reading Wing's souce code and no guarantee is made that these will remain unchanged or will change only in a backward compatible manner.

7.5. Advanced Scripting

While simple scripts can generally be developed from example using only the Wing IDE binary distribution, more advanced scripts require Wing to be run from the source code distribution, usually as a debug process being controlled by another copy of Wing IDE.

This provides not only more complete access to the source code for scripts that reach through the API into Wing internals, but also more complete support for debugging the scripts as they are developed.

To obtain Wing's source code, you must have a valid license to Wing IDE Professional or higher and must fill out and submit a non-disclosure agreement. Once this is done, you will be provided with access to the source code and more information on working with Wing IDE's sources.

Example

For an example of an advanced script that adds a tool panel to the IDE's interface, see `templating.py` in the `scripts` directory inside the Wing IDE installation.

How Script Reloading Works

Advanced scripters working outside of the API defined in `wingapi.py` should note that Wing only clears code objects registered through the API. For example, a script-added

timeout (using `CAPIApplication.InstallTimeout()` method) will be removed and re-added automatically during reload, but a tool panel added using Wing internals will need to be removed and re-added before it updates to run on altered script code. In some cases, when object references from a script file are installed into Wing's internals, it will be necessary to restart Wing IDE.

Here is how reloading works:

1) All currently loaded script files are watched so that saving the file from an editor will cause Wing to initiate reload after it has been saved.

2) When a file changes, all scripts in its directory will be reloaded.

3) Wing removes all old scripts from the command registry and registers any timeouts set with `CAPIApplication.InstallTimeout()`.

4) Next `imp.find_module` is used to locate the module by name.

5) Then the module is removed from `sys.modules` and reloaded using `imp.find_module` and a module name that prepends `internal_script_` to the module name (in order to avoid conflicting with other modules loaded by the IDE).

6) If module load fails (for example, due to a syntax error), any timeouts registered by the module during partial load are removed and the module is removed from `sys.modules`.

7) If the module contains `_ignore_scripts`, then its timeouts (if any) are removed and scripts in the file are ignored.

8) Otherwise, Wing adds all the scripts in the module to the command registry and loads any sub-modules if the module is a package with `__init__.py`.

Note that reloading is by design slightly different than Python's builtin `reload()` function: Any old top-level symbols are blown away rather than being retained. This places some limits on what can be done with global data: For example, storing a database connection will require re-establishing the connection each time the script is reloaded.

Preferences Reference

This chapter documents the entire set of available preferences for Wing IDE. Most preferences can be set from the **Preferences GUI** but some users may wish to build preference files manually to control different instances of Wing IDE (see details in **Preferences Customization**).

User Interface

Display Theme

Configures the overall display style, or theme, used by Wing IDE. Additional GTK2 themes may be downloaded from http://art.gnome.org/themes and placed into WINGHOME/bin/gtk-bin/share/themes or USER_SETTINGS_DIR/themes. These will be added to the choices below. However, only the pixbuf, metal, and redmond95 theme engines are supported.

Internal Name:

```
gui.display-theme
```

Data Specification:

```
[H2O-gtk2-Saphire, H2O-gtk2-Emerald, H2O-gtk2-Amber, AluminumAlloy-
Toxic, Redmond95, Smooth-2000, H2O-gtk2-Amythist, High-
ContrastLargePrint, AluminumAlloy-Cryogenic, HighCon-
trast, AluminumAlloy-Volcanic, LowContrast, LargePrint, High-
ContrastLargePrintInverse, AluminumAlloy-Smog, HighCon-
trastInverse, Smokey-Blue, Glider, Smooth-Sea-Ice, De-
fault, Glossy P, Redmond, None, Smooth-Retro, Smooth-
Desert, H2O-gtk2-Ruby, LowContrastLargePrint, Black-
Background, GnuBubble]
```

Default Value:

```
None
```

Display Language

The language to use for the user interface. Either the default for this system, or set to a specific supported language.

Internal Name:

```
main.display-language
```

Data Specification:

```
[None, de, en, fr]
```

Default Value:

```
None
```

Display Font/Size

The base font and size to use for the user interface's menus and labels

Internal Name:

```
gui.default-font
```

Data Specification:

```
[None or <type str>]
```

Default Value:

```
None
```

Source Code Font/Size

The base font and size to use for the source code editor, Python Shell, Debug Probe, Source Assistant, and other tools that display source code.

Internal Name:

```
edit.default-font
```

Data Specification:

```
[None or <type str>]
```

Default Value:

```
None
```

Use System Gtk

Use the system wide gtk library (requires gtk 2.2 or later). Wing comes with its own private copy of the gtk libraries for which it is built and tested. Use the system gtk option to better integrate with the gnome or other desktop environment, however on some systems this may result in random crashing or other bugs resulting from binary incompatibilities in library versions. This preference may be overridden on the command line with the --system-gtk and --private-gtk command line options.

Internal Name:

```
gui.use-system-gtk
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
False
```

- **Layout**

Windowing Policy

Policy to use for window creation: combined-window mode places toolboxes into editor windows, separate-toolbox-window mode creates separate tool box windows, and one-window-per-editor mode also creates a new window for each editor.

Internal Name:

```
gui.windowing-policy
```

Data Specification:

```
[combined-window, one-window-per-editor, separate-toolbox-
window]
```

Default Value:

```
combined-window
```

First Tool Box Location

Configures location of the tall panel area in the main display Window.

Internal Name:

```
gui.tall-panel-location
```

Data Specification:

```
[right, left]
```

Default Value:

```
right
```

Second Tool Box Location

Configures location of the wide panel area in the main display Window.

Internal Name:

```
gui.wide-panel-location
```

Data Specification:

```
[top, bottom]
```

Default Value:

```
bottom
```

Show Editor Notebook Tabs

Controls whether or not Wing shows notebook tabs for switching between editors. When false, a popup menu is used instead.

Internal Name:

```
gui.use-notebook-editors
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Enable Tooltips

Controls whether or not tooltips containing help are shown when the mouse hovers over areas of the user interface.

Internal Name:

```
gui.enable-tooltips
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

- **Toolbars**

Show Toolbar

Whether toolbar is shown in any window.

Internal Name:

```
gui.show-toolbar
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Toolbar Size

Sets size of the toolbar icons. One of “small”, “medium”, “large”, “xlarge”, or use “default” to select the system-wide settings.

Internal Name:

```
gui.toolbar-icon-size
```

Data Specification:

```
[medium, default, xlarge, text-height, large, small]
```

Default Value:

```
small
```

Toolbar Style

Select style of toolbar icons to use. One of “icon-only”, “text-only”, “text-below”, “text-right”, or use “default” to select the system-wide settings.

Internal Name:

```
gui.toolbar-icon-style
```

Data Specification:

```
[medium, default, xlarge, text-height, large, small]
```

Default Value:

```
text-right
```

- **Colors**

Text Selection Color

The color used to indicate the current text selection on editable text.

Internal Name:

```
gui.text-selection-color
```

Data Specification:

```
[tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]
```

Default Value:

```
(253, 253, 104)
```

Source Code Background

Background color to use on the source editor, Python Shell, Debug Probe, Source Assistant, and other tools that display source code. Foreground colors for text may be altered automatically to make them stand out on the selected background color.

Internal Name:

```
edit.background-color
```

Data Specification:

```
[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]
```

Default Value:

```
None
```

Debugger Run Marker Color

The color of the text hilight used for the run position during debugging

Internal Name:

```
debug.run-marker-color
```

Data Specification:

```
[tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]
```

Default Value:

```
(255, 163, 163)
```

Syntax Formatting

Formatting options for syntax coloring in editors. Colors are relative to a white background and will be transformed if the background color is set to a color other than white.

Internal Name:

```
.edit.syntax-formatting
```

Data Specification:

```
[dict; keys: <type str>, values: [dict; keys: [italic, back, fore, bold], va
ues: [one of: None, <type str>, <boolean: 0 or 1>]]]
```

Default Value:

```
{}
```

- **Keyboard**

Personality

Selects editor personality

Internal Name:

```
edit.personality
```

Data Specification:

```
[vi, visualstudio, emacs, brief, normal]
```

Default Value:

```
normal
```

Tab Key Action

Defines the action that the tab key has in files by type when it is bound to the tab key command. Possible actions are "Indent To Match" to indent the current line or selected lines to match the computed indent level for this context, "Increase Indent" to increase indentation one level, or "Insert Tab Character" to insert a Tab character (chr(9)).

Internal Name:

```
edit.tab-key-action
```

Data Specification:

```
[dict; keys: <type str>, values: <type str>]
```

Default Value:

```
{'*': '--default--', 'text/x-python': '--default--'}
```

Custom Key Bindings

Override key bindings in the keymap. To enter the key, place focus on the entry area and type the key combination desired. The command is one of those documented in the user manual's Command Reference, or the name of any user scripts that have been loaded into the IDE.

Internal Name:

```
gui.keymap-override
```

Data Specification:

```
[dict; keys: <type str>, values: <type str>]
```

Default Value:

```
{}
```

Typing Group Timeout

Sets the timeout in seconds to use for typing, after which keys pressed are considered a separate group of characters. This is used for typing-to-select on lists and in other GUI areas. Before the timeout subsequent keys are added to previous ones to refine the selection during keyboard navigation.

Internal Name:

```
gui.typing-group-timeout
```

Data Specification:

```
<type float>, <type int>
```

Default Value:

```
1
```

VI Mode Ctrl-C/X/V

Controls the behavior of the Ctrl-X/C/V key bindings in vi mode. Either always use these for cut/copy/paste, use them for vi native actions such as initiate-numeric-repeat and start-select-block, or use the default by system (clipboard on win32 and OS X, and other commands elsewhere).

Internal Name:

```
vi-mode.clipboard-bindings
```

Data Specification:

```
[other, clipboard, system-default]
```

Default Value:

```
system-default
```

- **Other**

Show Splash Screen

Controls whether or not the splash screen is shown at startup.

Internal Name:

```
main.show-splash-screen
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Case Sensitive Sorting

Controls whether names are sorted case sensitively (with all caps preceding small letters) or case insensitively

Internal Name:

```
gui.sort-case-sensitive
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Auto-Show Bug Report Dialog

Whether the error bug reporting dialog (also avaiable from the Help menu) is shown automatically when an unexpected exception is encountered inside Wing IDE.

Internal Name:

```
gui.show-report-error-dialog
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
False
```

Auto-check for Product Updates

Automatically attempt to connect to wingware.com to check for updates once every day after Wing is started.

Internal Name:

```
main.auto-check-updates
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

- **Advanced**

Display Area

Rectangle to use for the IDE work area on screen. All windows open within this area. Format is (x, y, width, height), or use None for full screen.

Internal Name:

```
gui.work-area-rect
```

Data Specification:

```
[None or [tuple length 4 of: <type int>, <type int>, <type int>, <type int>]]
```

Default Value:

```
None
```

Max Error Log Size

The number of bytes at which the error log file (USER_SETTINGS_DIR/error-log) is truncated. This file can be sent to technical support to help diagnose problems with the IDE.

Internal Name:

```
main.max-error-log-size
```

Data Specification:

```
[from 10000 to 10000000]
```

Default Value:

```
100000
```

Key Map File

Defines location of the keymap override file. Use None for default according to configured editor personality. See the Wing IDE Manual for details on building your keymap override file -- in general this is used only in development or debugging keymaps; use the keymap-override preference instead for better tracking across Wing versions.

Internal Name:

```
gui.keymap
```

Data Specification:

```
[None or <type str>]
```

Default Value:

```
None
```

Messages

Controls the format and verbosity of messages shown to the user for each message domain in the message area. Each domain specifies the format (in Python 2.3 logging.Formatter format), and the minimum logging level that should be shown in the display. If a message domain is left unspecified, then the parent domain settings are used instead ("" is the parent of all domains).

Internal Name:

```
gui.message-config
```

Data Specification:

```
[dict; keys: [search, debugger, analysis, general, project, ed-
itor, scripts, browser], values: [tuple length 3 of: <type str>, [0, 40, 30]
```

Default Value:

```
{'': ('%(message)s', 0, 100000)}
```

Document Text Styles

Defines text styles used in data and document display. Each style is specified as a list of (name, value) tuples. The names and values must be valid Pango text attribute names and values. To set default values that apply to all styles, use the "default" style name (for example adding ("size", 14) changes default dislpay size to 14 points. Note that size of menus, buttons, labels, and other basic GUI elements are set using system-wide theme configuration and not from this preference. The source editor is also configured separately.

Internal Name:

```
main.text-styles
```

Data Specification:

```
[dict; keys: [one of: <type str>, [admonition-title, dan-
ger, footnote, citation, admonition, calltip-doc, title-
```

```
4, calltip-strong, caution, title-3, title-0, title-
1, image-link, calltip-type, calltip-poc, hint, calltip-arg-
current, tip, literal, note, field, emphasis, title-2, calltip-
class-symbol, attention, calltip-def-symbol, link, strong, marked-
list-items, calltip-def, list-items, default, docinfo-
header, transition, calltip-arg, caption, warning, er-
ror, navigation-link, navigation]], values: [tuple of: [one of: [tu-
ple length 2 of: [foreground], [None or <type str>]], [tu-
ple length 2 of: [style], [None or [oblique, italic, nor-
mal]]], [tuple length 2 of: [justification], [None or [right, fill, cen-
ter, left]]], [tuple length 2 of: [font_desc], [None or <type str>]], [tu-
ple length 2 of: [weight], [None or [one of: <type int>, [heavy, bold, ul-
trabold, normal, light, ultralight]]]], [tuple length 2 of: [right_margin], [None or
ple length 2 of: [stretch], [None or [condensed, ex-
panded, normal, semicondensed, extracondensed, extraex-
panded, semiexpanded, ultracondensed, ultraexpanded]]], [tu-
ple length 2 of: [strikethrough], [None or <boolean: 0 or 1>]], [tu-
ple length 2 of: [rise], [None or [from -100000 to 100000]]], [tu-
ple length 2 of: [variant], [None or [smallcaps, nor-
mal]]], [tuple length 2 of: [underline], [None or [double, sin-
gle, low, none]]], [tuple length 2 of: [ypad], [None or [1]]], [tu-
ple length 2 of: [background], [None or <type str>]], [tu-
ple length 2 of: [indent], [None or [1]]], [tuple length 2 of: [left_margin], [None
ple length 2 of: [font_family], [None or <type str>]], [tu-
ple length 2 of: [xpad], [None or [1]]], [tuple length 2 of: [size], [None or [one o
large, xx-large, large, small, xx-small, x-small]]]]]]]
```

Default Value:

```
{'calltip-strong': (('font_family', 'sans'), ('weight', 'bold'), ('fore-
ground', '#000066')), 'danger': (('background', '#ffffdd'),), 'foot-
note': (('weight', 'bold'),), 'navigation-link': (('fore-
ground', '#909090'), ('style', 'italic'), ('weight', 'bold')), 'ci-
tation': (('weight', 'bold'),), 'admonition': (), 'list-
items': (('xpad', '1'), ('ypad', '1')), 'title-4': (('size', 'small'), ('un-
derline', 'single'), ('foreground', '#000066')), 'warn-
ing': (('background', '#ffffdd'),), 'caution': (('back-
ground', '#ffffdd'),), 'title-3': (('size', 'small'), ('weight', 'bold'), ('fore-
ground', '#000066')), 'title-0': (('size', 'xx-large'), ('weight', 'bold'), ('fore-
ground', '#000066')), 'title-1': (('size', 'large'), ('weight', 'bold'), ('fore-
ground', '#000066')), 'image-link': (), 'calltip-type': (('font_family', 'sans'),),
poc': (('font_family', 'sans'),), 'hint': (('background', '#ffffdd'),), 'admonition-
title': (('weight', 'bold'),), 'tip': (('background', '#ffffdd'),), 'lit-
```

```
eral': (('foreground', '#227722'), ('weight', 'bold')), 'note': (), 'field':
phasis': (('style', 'italic'),), 'calltip-class-symbol': (('font_family', 's
ground', '#0000ff')), 'attention': (('background', '#ddddff'),), 'calltip-
def-symbol': (('font_family', 'sans'), ('weight', 'bold'), ('fore-
ground', '#007f7f')), 'link': (('underline', 'single'), ('fore-
ground', '#3333ff')), 'strong': (('weight', 'bold'), ('fore-
ground', '#000066')), 'marked-list-items': (('weight', 'bold'), ('fore-
ground', '#ff3333')), 'calltip-def': (('font_family', 'sans'), ('weight', 'b
ground', '#00007f')), 'calltip-doc': (('font_family', 'sans'),), 'de-
fault': (), 'docinfo-header': (('weight', 'bold'),), 'transi-
tion': (('justification', 'left'),), 'calltip-arg': (('font_family', 'sans')
arg-current': (('font_family', 'sans'), ('background', '#ff-
bbbb')), 'caption': (('style', 'italic'),), 'error': (('back-
ground', '#ffdddd'),), 'title-2': (('size', 'medium'), ('weight', 'bold'), (
ground', '#000066')), 'navigation': (('foreground', '#909090'), ('style', 'i
```

Files

Default Directory Policy

Defines how Wing determines the starting directory to use when prompting for a file name: Either based on location of the resource at current focus, location of the current project, the last directory visited for file selection, the current directory at startup (or selected since), or always the specific fixed directory entered here.

Internal Name:

```
main.start-dir-policy
```

Data Specification:

```
[tuple length 2 of: [current-project, current-directory, recent-
directory, current-focus, selected-directory], <type str>]
```

Default Value:

```
('current-focus', '')
```

Title Style

Format used for titles of source files: Use “basename” to display just the file name, “prepend-relative” to use partial relative path from the project file location, “append-relative” to append partial relative path from project file location after the base file name, “prepend-fullpath” to use full path, or “append-fullpath” to append fullpath after the based file name.

Internal Name:

```
gui.source-title-style
```

Data Specification:

```
[append-relative, basename, prepend-fullpath, append-
fullpath, prepend-relative]
```

Default Value:

```
append-relative
```

Default Encoding

The default encoding to use for text files opened in the source editor and other tools, when an encoding for that file cannot be determined by reading the file. Other encodings may also be tried. This also sets the encoding to use for newly created files.

Internal Name:

```
edit.default-encoding
```

Data Specification:

```
[None or [Central and Eastern European iso8859-2, Japanese iso-
2022-jp-2004, Hebrew cp856, Japanese euc-jp, Vietnamese cp1258, Greek cp1253, Baltic
guages cp1257, Korean johab, Western European cp1252, Baltic Lan-
guages cp775, Japanese iso-2022-jp-ext, Korean iso-2022-
kr, Icelandic cp861, Hebrew cp424, Cyrillic Languages cp1251, Turk-
ish iso8859-9, Unicode (UTF-16, little endian) utf-16-le, West-
ern European cp500, Chinese (PRC) gb18030, Greek cp875, Ara-
bic cp864, Icelandic mac-iceland, Chinese (PRC) gbk, Turk-
ish mac-turkish, Greek iso8859-7, Baltic Languages iso8859-
13, Cyrillic Languages mac-cyrillic, Greek cp869, Japanese iso-
```

```
2022-jp-1, Central and Eastern European cp852, Japanese iso-
2022-jp-2, Chinese (ROC) big5, Urdu cp1006, Hebrew iso8859-
8, Celtic Languages iso8859-14, Thai cp874, Cyrillic Lan-
guages cp855, Western European iso8859-15, Greek mac-
greek, Ukrainian koi8-u, Hebrew cp1255, Danish, Norwe-
gian cp865, Cyrillic Languages iso8859-5, Turkish cp1026, West-
ern European mac-roman, Western European cp1140, Chi-
nese (PRC) hz, Japanese shift-jisx0213, Portuguese cp860, Chi-
nese (ROC) cp950, US, Canada, and Others cp037, Japanese shift-
jis-2004, Turkish cp1254, Japanese iso-2022-jp-3, He-
brew cp862, Western European latin-1, Japanese euc-jisx0213, Uni-
code (UTF-16, big endian) utf-16-be, Japanese euc-jis-
2004, None, Central and Eastern European cp1250, Baltic Lan-
guages iso8859-4, English ascii, Japanese shift-jis, Ara-
bic iso8859-6, Canadian English/French cp863, System de-
fault (ISO-8859-1), Russian koi8-r, Japanese iso-2022-jp, Uni-
code (UTF-8) utf-8, Greek cp737, Nordic Languages iso8859-
10, Central and Eastern European mac-latin2, Chinese (PRC) gb2312, Uni-
code (UTF-7) utf-7, Arabic cp1256, Chinese (PRC) big5hkscs, West-
ern European cp850, Esperanto and Maltese iso8859-3, Turk-
ish cp857, Korean cp949, US, Australia, New Zealand, S. Africa cp437, Uni-
code (UTF-16) utf-16, Japanese cp932]]
```

Default Value:

```
None
```

New File EOL

Default end-of-line to use: One of “lf”, “cr”, or “crlf” for each entry. Note that Wing matches existing line endings in non-blank files and uses this preference only when a file contains no end-of-line characters.

Internal Name:

```
edit.new-file-eol-style
```

Data Specification:

```
[lf, cr, crlf]
```

Default Value:

```
lf
```

New File Extension

Default file extension for newly created files

Internal Name:

```
edit.new-file-extension
```

Data Specification:

```
<type str>
```

Default Value:

```
.py
```

Max Recent Items

Maximum number of items to display in the Recent menus.

Internal Name:

```
gui.max-recent-files
```

Data Specification:

```
[from 3 to 200]
```

Default Value:

```
20
```

Always Use Full Path in Tooltips

Set to True to always show the full path of a file name in the tooltips shown from the editor tabs and file selection menus. When False, the configured Source Title Style is used instead.

Internal Name:

```
gui.full-path-in-tooltips
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
True
```

- **File Types**

Extra File Types

This is a map from file extension or wildcard to mime type. It adds additional file type mappings to those built into Wing IDE. File extensions can be specified alone without dot or wildcard, for example "xcf" or using wildcards containing "*" and/or "?", for example "Makefile*". The mime type to use for Python files is "text/x-python".

Internal Name:

```
main.extra-mime-types
```

Data Specification:

```
[dict; keys: <type str>, values: [text/x-sql, text/x-
pov, text/x-ave, text/x-pl-sql, text/x-bash, text/x-lua-
source, text/x-eiffel, text/x-vxml, text/xml, text/x-
errorlist, text/x-php-source, text/x-dos-batch, text/x-
bullant, text/x-baan, text/x-python, text/x-nncrontab, text/x-
mmixal, text/postscript, text/x-javascript, text/x-fortran, text/x-
xcode, text/x-escript, text/x-lisp, text/x-makefile, text/x-
diff, text/x-ms-idl, text/x-cpp-source, text/x-asm, text/x-
ruby, text/x-ada, text/x-nsis, text/x-idl, text/x-scriptol, text/x-
zope-pt, text/x-perl, text/x-java-source, text/x-docbook, text/x-
rc, text/x-c-source, text/plain, text/x-lout, text/x-
matlab, text/html, application/x-tex, text/x-tcl, text/x-vb-
source, text/x-pascal, text/x-yaml, text/x-conf, text/x-ms-
makefile, text/x-properties, text/css]]
```

Default Value:

```
{}
```

File Sets

Defines file sets by specifying filters to apply to file names for inclusion and exclusion from a larger set (such as scanned disk files or all project files).

Each file set is named and contains one list of inclusion patterns and one list of exclusion patterns. The patterns can be a wildcard on the file name, wildcard on a directory name, or a mime type name.

> Only a single pattern needs to be matched for inclusion or exclusion. Exclusion patterns take precedence over inclusion patterns, so any match on an exclusion pattern will always exclude a file from the set. File sets are used in constraining search, adding project files, and for other operations on collections of files.

Internal Name:

```
main.file-sets
```

Data Specification:

```
[dict; keys: <type str>, values: [tuple length 2 of: [tu-
ple of: [tuple length 2 of: [wildcard-filename, wildcard-
directory, mime-type], <type str>]], [tuple of: [tu-
ple length 2 of: [wildcard-filename, wildcard-directory, mime-
type], <type str>]]]]
```

Default Value:

```
{u'All Source Files': ((), (('wildcard-filename', '*.o'), ('wildcard-
filename', '*.obj'), ('wildcard-filename', '*.a'), ('wildcard-
filename', '*.lib'), ('wildcard-filename', '*.so'), ('wildcard-
filename', '*.dll'), ('wildcard-filename', '*.exe'), ('wildcard-
filename', '*.ilk'), ('wildcard-filename', '*.pdb'), ('wildcard-
filename', '*.pyc'), ('wildcard-filename', '*.pyo'), ('wildcard-
filename', '*.pyd'), ('wildcard-filename', 'core'), ('wildcard-
filename', '*.bak'), ('wildcard-filename', '*.tmp'), ('wildcard-
filename', '*.temp'), ('wildcard-filename', '*-old'), ('wildcard-
filename', '*.old'), ('wildcard-filename', '*.wpr'), ('wildcard-
```

```
filename', '*.wpu'), ('wildcard-filename', '*.zip'), ('wildcard-
filename', '*.tgz'), ('wildcard-filename', '*.tar.gz'), ('wildcard-
filename', '*~'), ('wildcard-filename', '#*#'), ('wildcard-
filename', '.#*'), ('wildcard-filename', '*.svn-base'), ('wildcard-
directory', 'CVS'), ('wildcard-directory', '.svn'), ('wildcard-
directory', '_svn'), ('wildcard-directory', '.xvpics'))), u'HTML and XML Fil
type', 'text/html'), ('mime-type', 'text/xml'), ('mime-
type', 'text/x-zope-pt')), (('wildcard-filename', '*~'), ('wildcard-
filename', '#*#'), ('wildcard-filename', '.#*'), ('wildcard-
filename', '*.svn-base'), ('wildcard-directory', 'CVS'), ('wildcard-
directory', '.svn'), ('wildcard-directory', '_svn'), ('wildcard-
directory', '.xvpics'))), u'C/C++ Files': ((('mime-type', 'text/x-
c-source'), ('mime-type', 'text/x-cpp-source')), (('wildcard-
filename', '*~'), ('wildcard-filename', '#*#'), ('wildcard-
filename', '.#*'), ('wildcard-filename', '*.svn-base'), ('wildcard-
directory', 'CVS'), ('wildcard-directory', '.svn'), ('wildcard-
directory', '_svn'), ('wildcard-directory', '.xvpics'))), u'Python Files': (
type', 'text/x-python'),), (('wildcard-filename', '*~'), ('wildcard-
filename', '#*#'), ('wildcard-filename', '.#*'), ('wildcard-
filename', '*.svn-base'), ('wildcard-directory', 'CVS'), ('wildcard-
directory', '.svn'), ('wildcard-directory', '_svn'), ('wildcard-
directory', '.xvpics')))}
```

- **Reloading**

External Check Freq

Time in seconds indicating the frequency with which the IDE should check the disk for files that have changed externally. Set to 0 to disable entirely.

Internal Name:

```
cache.external-check-freq
```

Data Specification:

```
<type float>, <type int>
```

Default Value:

```
5
```

Reload when Unchanged

Selects action to perform on files found to be externally changed but unaltered within the IDE. One of "auto-reload" to automatically reload these files, "request-reload" to ask via a dialog box upon detection, "edit-reload" to ask only if the unchanged file is edited within the IDE subsequently, or "never-reload" to ignore external changes (although you will still be warned if you try to save over an externally changed file)

Internal Name:

```
cache.unchanged-reload-policy
```

Data Specification:

```
[never-reload, auto-reload, request-reload, edit-reload]
```

Default Value:

```
auto-reload
```

Reload when Changed

Selects action to perform on files found to be externally changed and that also have been altered in the IDE. One of "request-reload" to ask via a dialog box upon detection, "edit-reload" to ask if the file is edited further, or "never-reload" to ignore external changes (although you will always be warned if you try to save over an externally changed file)

Internal Name:

```
cache.changed-reload-policy
```

Data Specification:

```
[never-reload, request-reload, edit-reload]
```

Default Value:

```
request-reload
```

- **Projects**

Auto-reopen Last Project

Controls whether most recent project is reopened at startup, in the absence of any other project on the command line.

Internal Name:

```
main.auto-reopen-last-project
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Close Files with Project

Controls whether any files open in an editor are also closed when a project file is closed

Internal Name:

```
proj.close-also-windows
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Default Type

Controls the type of project file that is written by default for new projects: “normal” for regular single-file format with extension .wpr, and “shared” for split format where the .wpr file contains shared project info that can be checked into a shared revision control repository and the .wpu file contains user-specific information such as location of breakpoints. This is useful to avoid revision control wars on a project with multiple developers.

Internal Name:

```
proj.file-type
```

Data Specification:

```
[shared, normal]
```

Default Value:

```
normal
```

Auto-Adding

Controls whether files are added to the current project automatically. Either add all files that get saved to disk while the project is open, add only newly created files, or don't auto-add any files.

Internal Name:

```
proj.auto-add-policy
```

Data Specification:

```
[all-saved, all-new, never]
```

Default Value:

```
never
```

Open Projects as Text

Controls whether project files are opened as project or as text when opened from the File menu. This does not affect opening from the Project menu.

Internal Name:

```
gui.open-projects-as-text
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

- **External Display**

File Display Commands

Linux only: The commands used to display or edit local disk files selected from the Help menu or project files selected for external display. This is a map from mime type to a list of display commands; each display command is tried in order of the list until one works. The mime type "*" can be used to set a generic viewer, such as a web browser. Use %s to place the file name on the command lines. If unspecified then Wing will use the configured URL viewer in the environment (specified by BROWSER environment variable or by searching the path for common browsers). On Windows and OS X, the system-wide configured default viewer for the file type is used instead so this preference is ignored.

Internal Name:

```
gui.file-display-cmds
```

Data Specification:

```
[dict; keys: <type str>, values: [list of: <type str>]]
```

Default Value:

```
{}
```

Url Display Commands

Linux only: The commands used to display URLs. This is a map from protocol type to a list of display commands; each display command is tried in order of the list until one works. The protocol "*" can be used to set a generic viewer, such as a multi-protocol web browser. Use %s to place the URL on the command lines. If unspecified then Wing will use the configured URL viewer in the environment (specified by BROWSER environment variable or by searching the path for common browsers). On Windows and OS X, the system-wide configured default web browser is used instead so this preference is ignored.

Internal Name:

```
gui.url-display-cmds
```

Data Specification:

```
[dict; keys: <type str>, values: [list of: <type str>]]
```

Default Value:

```
{}
```

Editor

Line Number Column Width

Width of the line number column (0 to hide). If value is less than 10, the value is the # of digits and if it is 10 or more, it is the number of pixels

Internal Name:

```
edit.lineno-column-width
```

Data Specification:

```
<type int>
```

Default Value:

```
0
```

Caret Width

Width of the blinking insertion caret on the editor, in pixels. Currently limited to a value between 1 and 3.

Internal Name:

```
edit.caret-width
```

Data Specification:

```
[from 1 to 3]
```

Default Value:

```
1
```

Show Whitespace

Set to true to show whitespace with visible characters by default

Internal Name:

```
edit.show-whitespace
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Show EOL

Set to true to show end-of-line with visible characters by default

Internal Name:

```
edit.show-eol
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Input Method

Input method used for typing characters. This is important primarily for non-Western European languages.

Internal Name:

```
edit.gtk-input-method
```

Data Specification:

```
[]
```

Default Value:

```
default
```

Split Reuse Policy

Policy for reusing splits in editors when new files are opened: Either open in current split or open in an adjacent split. This only has an effect when more than one editor split is visible.

Internal Name:

```
gui.split-reuse-policy
```

Data Specification:

```
[current, adjacent]
```

Default Value:

```
current
```

- **Indentation**

Use Indent Analysis

Select when to use indent analysis (examination of current file contents) in order to determine tab size and indent size. Either always in all files, only in Python files, or never.

Internal Name:

```
edit.use-indent-analysis
```

Data Specification:

```
[always, never, python-only]
```

Default Value:

```
always
```

Default Tab Size

Set size of tabs, in spaces, used in new files. Note that in Python files that contain mixed space and tab indentation, tab size is always forced to 8 spaces. Use the Indentation Manager to alter indentation in existing files.

Internal Name:

```
edit.tab-size
```

Data Specification:

```
[from 0 to 80]
```

Default Value:

```
8
```

Default Indent Size

Sets size of an indent, in spaces, used in new files. This is overridden in non-empty files, according to the actual contents of the file. In files with tab-only indentation, this value may be modified so it is a multiple of the configured tab size. Use the Indentation Manager to alter indentation in existing files.

Internal Name:

```
edit.indent-size
```

Data Specification:

```
[from 0 to 80]
```

Default Value:

```
4
```

Default Indent Style

Set the style of indentation used in new files. This is overridden in non-empty files, according to the actual contents of the file. Use the Indentation Manager to alter indentation in existing files. Indentation style choices are "tabs-only" for tabs only, "spaces-only" for spaces only, or "mixed" to use a tab whenever tab-size spaces have been seen

Internal Name:

```
edit.indent-style
```

Data Specification:

```
[mixed, spaces-only, tabs-only]
```

Default Value:

```
spaces-only
```

Auto Indent

Controls when Wing automatically indents when return or enter is typed.

Internal Name:

```
edit.auto-indent
```

Data Specification:

```
[0, 1, blank-only]
```

Default Value:

```
1
```

Show Indent Guides

Set to true to show indent guides by default

Internal Name:

```
edit.show-indent-guides
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Show Python Indent Warnings

Set to cause Wing to show warnings when opening a Python file with potentially problematic indentation (either inconsistent and possibly confusing indentation, a mix of indent styles in a single file, or mixed tab and space indentation (which is not recommended for Python).

Internal Name:

```
edit.show-python-indent-warnings
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Show Override Warnings

Show indent mismatch warning dialog when user selects an indent style that is incompatible with existing file content. This only applies to non-Python files since Wing disallows overriding the indent style in all Python files.

Internal Name:

```
edit.show-non-py-indent-warning
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
True
```

- **Line Wrapping**

Wrap Long Lines

Set to true to wrap long source lines on the editor display.

Internal Name:

```
edit.wrap-lines
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Edge Markers

Tuple that defines how edge markers are shown: (mode, column, color) where mode is 0 to turn off markers, 1 to show a line, or 2 to highlight text that extends past the edge; column is the column at which to draw the marker, if on; and color is the color for the marker (r,g,b) tuple with values from 0x00 to 0xff: (0xff,0xff,0xff) is white.

Internal Name:

```
edit.show-edge-markers
```

Data Specification:

```
[tuple length 3 of: [0, 1, 2], [from 0 to 10000], [tu-
ple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]
```

Default Value:

```
(0, 80, (251, 8, 8))
```

Reformatting Wrap Column

Column at which text should be wrapped by commands that automatically rearrange text

Internal Name:

```
edit.text-wrap-column
```

Data Specification:

```
<type int>
```

Default Value:

```
77
```

- **Folding**

Enable Folding

Set to true to enable structural folding on source, false to disable

Internal Name:

```
edit.enable-folding
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Line Mode

Set to “above-expanded”, “below-expanded”, “above-collapsed”, “below-collapsed”, or “none” to indicate where fold lines are shown and whether they are above or below the line where the fold point is located.

Internal Name:

```
edit.fold-line-mode
```

Data Specification:

```
[above-collapsed, above-expanded, none, below-collapsed, below-
expanded]
```

Default Value:

```
below-collapsed
```

Indicator Style

Set to 0 to use arrow indicators, 1 to use plus/minus indicators, 2 to rounded tree indicators, and 3 to use square tree indicators.

Internal Name:

```
edit.fold-indicator-style
```

Data Specification:

```
[from 0 to 3]
```

Default Value:

```
1
```

- **Auto-completion**

Auto-show Completer

Controls whether or not the completer is shown automatically during typing. When disabled, it can still be shown on demand with the Show Completer item in the Source menu.

Internal Name:

```
edit.autocomplete-autoshow
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Auto-complete Delay

Delay in seconds from last key press to wait before the auto-completer is shown. If 0.0, the auto-completer is shown immediately.

Internal Name:

```
edit.autocomplete-delay
```

Data Specification:

```
<type int>, <type float>
```

Default Value:

```
0.0
```

Auto-complete Timeout

Timeout in seconds from last key press after which the auto-completer is automatically hidden. If 0.0, the auto-completer does not time out.

Internal Name:

```
edit.autocomplete-timeout
```

Data Specification:

```
<type int>, <type float>
```

Default Value:

```
0
```

Completion Keys

Controls which keys will enter selected completion value into the editor. Shift or Ctrl click to select multiple items.

Internal Name:

```
edit.autocomplete-keys
```

Data Specification:

```
[tuple of: [f1, f3, return, space, period, bracketleft, tab, f12, f10, paren-
left]]
```

Default Value:

```
('tab',)
```

Completion Mode

Selects how completion is done in the editor: Either insert the completion at the cursor, or replace any existing symbol with the new symbol.

Internal Name:

```
edit.autocomplete-mode
```

Data Specification:

```
[insert, replace]
```

Default Value:

```
insert
```

Case Insensitive Matching

Controls whether matching in the completer is case sensitive or not. The correct case is always used when a completion is chosen.

Internal Name:

```
edit.autocomplete-case-insensitive
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
True
```

- **Printing**

Print Font

(Posix only) Set the font name used to print Python files. One of Courier, Helvetica, or Times-Roman.

Internal Name:

```
edit.print-font
```

Data Specification:

```
[Times-Roman, Helvetica, Courier]
```

Default Value:

```
Courier
```

Print Size

(Posix only) Set the font size used to print Python files.

Internal Name:

```
edit.print-size
```

Data Specification:

```
[from 0 to 120]
```

Default Value:

```
10
```

Paper

(Posix only) Set the paper size for printing. One of Letter, Legal, A3, A4, A5, B4, or B5

Internal Name:

```
edit.print-paper
```

Data Specification:

```
[A3, A5, Legal, Letter, A4]
```

Default Value:

```
Letter
```

Print Spool Cmd

(Posix only) Sets the command used to spool output produced by Wing's printing facility. Format is text with embedded %s to indicate where the printed output file's name should be inserted. Set to None to use internal defaults. If the default is not working for you and your system does not accept PDF files for printing, try "pdf2ps %s - | kprinter --stdin". To rule out problems with buggy versions of kprinter, try "pdf2ps %s - | lpr" or simply "lpr %s" instead.

Internal Name:

```
edit.print-spool-cmd
```

Data Specification:

```
[one of: None, <type str>]
```

Default Value:

```
None
```

Print Python as Text

(Posix only) Set to true to print Python files faster but without syntax highlighting. Otherwise, the internal Python pretty printing facility is used.

Internal Name:

```
edit.print-python-as-text
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Text Print Cmd

(Posix only) Sets the command that is issued to print non-Python text files. Format is text with embedded %s to indicate where the printed file's name should be inserted

Internal Name:

```
edit.text-print-cmd
```

Data Specification:

```
<type str>
```

Default Value:

```
enscript -E %s
```

- **Advanced**

Auto Brace Match

Set to true to automatically match braces next to the cursor or as they are typed.

Internal Name:

```
edit.auto-brace-match
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Transient Threshold

Maximum number of transient (non-sticky) editors to keep open at one time, in addition to any that are visible on screen

Internal Name:

```
gui.max-non-sticky-editors
```

Data Specification:

```
<type int>
```

Default Value:

```
1
```

Selection Policy

This is a map from actions to policy for leaving a range selected after the action takes place. Possible actions are “indent-region”, “outdent-region”, “indent-to-match”, “comment-out-region”, and “uncomment-out-region”. Possible policies for each are “always-select”, which always leaves a selection, “retain-select” which leaves a selection only if there was one to begin with, and “never-select” which never leaves a selection.

Internal Name:

```
edit.select-policy
```

Data Specification:

```
[dict; keys: [(u'Indent Region', 'indent-region'), (u'Indent To Match', 'ind
to-match'), (u'Uncomment out Region', 'uncomment-out-
region'), (u'Outdent Region', 'outdent-region'), (u'Comment out Re-
gion', 'comment-out-region')], values: [(u'Never Se-
lect', 'never-select'), (u'Retain Select', 'retain-select'), (u'Always Se-
lect', 'always-select')]]
```

Default Value:

```
{'uncomment-out-region': 'retain-select', 'outdent-region': 'retain-
select', 'comment-out-region': 'retain-select', 'indent-
region': 'retain-select', 'indent-to-match': 'retain-select'}
```

Middle Mouse Paste

Paste text into the editor from the clipboard when the middle mouse button is pressed. Disabling this is mainly useful for wheel mice with a soft wheel that causes pasting of text before wheel scrolling starts.

Internal Name:

```
edit.middle-mouse-paste
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
True
```

Default Drag-n-Drop Action

Default drag-n-drop action. This is the default and can always be overridden by pressing shift or ctrl while dragging

Internal Name:

```
edit.default-drop-action
```

Data Specification:

```
[os-default, copy, move]
```

Default Value:

```
os-default
```

Debugger

Integer Display Mode

This sets the display style for integer values to one of “dec”, “hex”, or “oct”.

Internal Name:

```
debug.default-integer-mode
```

Data Specification:

```
[dec, hex, oct]
```

Default Value:

```
dec
```

Auto-save Files

Controls whether or not all edited files are autosaved before a debug run, or before a file or build process is executed.

Internal Name:

```
gui.auto-save-before-action
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Ignore Unsynchronized Files

Controls whether or not Wing ignores unsaved files before a debug run, or before a file or build process is executed.

Internal Name:

```
gui.ignore-unsaved-before-action
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Raise Source From Tools

Controls whether the debugger raises source files to indicate exception locations encountered when working in the Debug Probe, and other debugger tools.

Internal Name:

```
debug.raise-from-tools
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Default Watch Style

Sets the tracking style used when a value is double clicked in order to watch it: Use "symbolic" to track by symbolic name, "parent-ref" to track parent by object reference and attribute by name, and "ref" to track using an object reference directly to the value

Internal Name:

```
debug.default-watch-style
```

Data Specification:

```
[ref, parent-ref, symbolic]
```

Default Value:

```
symbolic
```

Line Threshold

Defines the character length threshold under which a value will always be shown on a single line, even if the value is a complex type like a list or map

Internal Name:

```
debug.line-threshold
```

Data Specification:

```
<type int>
```

Default Value:

```
65
```

- **Exceptions**

Report Exceptions

Controls how Wing reports exceptions that are raised by your debug process. By default, Wing tries to predict which exceptions are unhandled, and stops immediately when unhandled exceptions are raised. Alternatively, Wing can stop on all exceptions (even if handled) immediately when they are raised, or it can wait to report fatal exceptions as the debug process terminates. In the latter case Wing makes a best effort to stop before the debug process exits or at least to report the exception post-mortem, but one or both may fail if working with externally launched debug processes. In that case, we recommend using Immediately if Appear Unhandled exception reporting mode.

Internal Name:

```
debug.exception-mode
```

Data Specification:

```
[unhandled, always, never]
```

Default Value:

```
unhandled
```

Never Report

Names of builtin exceptions to never report, even if the exception is not handled. This list takes precedence over the always report list and the default reporting mode, but is not used if the exception reporting mode is set to always.

Internal Name:

```
debug.never-stop-exceptions
```

Data Specification:

```
[tuple of: <type str>]
```

Default Value:

```
['SystemExit', 'GeneratorExit']
```

Always Report

Names of builtin exceptions to (nearly) always report. These exceptions are not reported only if they explicitly caught by the specific subclass in the frame they are raised in.

Internal Name:

```
debug.always-stop-exceptions
```

Data Specification:

```
[tuple of: <type str>]
```

Default Value:

```
['AssertionError', 'NameError', 'UnboundLocalError']
```

- **I/O**

Use External Console

Selects whether to use the integrated I/O panel for debug process input/output or an external terminal window. Use an external window if your debug process depends on details of the command prompt environment for cursor movement, color text, etc.

Internal Name:

```
debug.external-console
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

External Console Waits on Exit

Set to true to leave up the console after normal program exit, or false to close the console right away in all cases. This is only relevant when running with an external native console instead of using the integrated debug I/O panel.

Internal Name:

```
debug.persist-console
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

External Consoles

A list of the xterm-compatible X windows terminal programs that are used with debug processes when running with an external console. Each is tried in turn until one is found to exist. If just the name is given, Wing will look for each first on the PATH and then in likely places. Specify the full path (starting with "/") to use a specific executable.

Internal Name:

```
debug.x-terminal
```

Data Specification:

```
[tuple of: <type str>]
```

Default Value:

```
('xterm', 'konsole', 'gnome-terminal', 'rxvt')
```

- **Data Filters**

Huge List Threshold

Defines the length threshold over which a list, map, or other complex type will be considered too large to show in the normal debugger. If this is set too large, the debugger will time out (see network-timeout preference)

Internal Name:

```
debug.huge-list-threshold
```

Data Specification:

```
<type int>
```

Default Value:

```
2000
```

Huge String Threshold

Defines the length over which a string is considered too large to fetch for display in the debugger. If this is set too large, the debugger will time out (see network-timeout preference).

Internal Name:

```
debug.huge-string-threshold
```

Data Specification:

```
<type int>
```

Default Value:

```
64000
```

Omit Types

Defines types for which values are never shown by the debugger.

Internal Name:

```
debug.omit-types
```

Data Specification:

```
[tuple of: <type str>]
```

Default Value:

```
('function', 'builtin_function_or_method', 'class', 'clas-
sobj', 'instance method', 'type', 'module', 'ufunc', 'wrap-
per_descriptor', 'method_descriptor', 'member_descriptor')
```

Omit Names

Defines variable/key names for which values are never shown by the debugger.

Internal Name:

```
debug.omit-names
```

Data Specification:

```
[tuple of: <type str>]
```

Default Value:

```
()
```

Do Not Expand

Defines types for which values should never be probed for contents. These are types that are known to crash when the debugger probes them because they contain buggy data value extraction code. These values are instead shown as an opaque value with hex object instance id.

Internal Name:

```
debug.no-probe-types
```

Data Specification:

```
[tuple of: <type str>]
```

Default Value:

```
('GdkColormap', 'IOBTree')
```

- **External/Remote**

Enable Passive Listen

Controls whether or not the debugger listens passively for connections from an externally launched program (false to disable; true to enable). This should be on when the debug program is not launched by the IDE (e.g., as with a CGI script).

Internal Name:

```
debug.passive-listen
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Allowed Hosts

Sets which hosts are allowed to connect to the debugger when it is listening passively for externally launched programs.

Internal Name:

```
debug.passive-hosts
```

Data Specification:

```
[tuple of: <type str>]
```

Default Value:

```
('127.0.0.1',)
```

Server Host

Determines the network interface on which the debugger listens for connections. This can be a symbolic name, an ip address, or left unspecified (use None) to indicate that the debugger should listen on all valid network interfaces on the machine. Note that when a debug session is launched from within the IDE (with the Run button), it always connects from the loopback interface (127.0.0.1)

Internal Name:

```
debug.network-server
```

Data Specification:

```
[None or <type str>]
```

Default Value:

```
None
```

Server Port

Determines the TCP/IP port on which the IDE will listen for the connection from the debug process. This needs to be unique for each developer working on a given host. The debug process, if launched from outside of the IDE, needs to be told the value specified here using kWingHostPort inside wingdbstub.py or by WINGDB_HOSTPORT environment variable before importing wingdbstub in the debug process.

Internal Name:

```
debug.network-port
```

Data Specification:

```
[from 0 to 65535]
```

Default Value:

```
50005
```

Location Map

Defines a mapping between the remote and local locations of files for host-to-host debugging. Each mapping key is the ip address of the remote location and the mapping values are arrays of tuples where each tuple is a (remote_prefix, local_prefix) pair. This should be used when files on the remote host are updated via ftp, NFS, Samba, or other method from master copies on the local host, but the full path file system locations on the local and remote hosts do not match.

Internal Name:

```
debug.location-map
```

Data Specification:

```
[dict; keys: <type str>, values: [None or [list of: [tu-
ple length 2 of: <type str>, <type str>]]]]
```

Default Value:

```
{'127.0.0.1': None}
```

Kill Externally Launched

Enable or disable the Kill command for debug processes that were launched from outside of the IDE

Internal Name:

```
debug.enable-kill-external
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Common Attach Hosts

List of host/port combinations that should be included by default in the attach request list shown with Attach to Process in the Debug menu, in addition to those that are registered at runtime. These are used primarily with externally launched debug processes, since Wing automatically shows IDE-launched processes for attach when appropriate. This value corresponds with kAttachPort configured in wingdbstub.py or by WINGDB_ATTACHPORT environment variable before importing wingdbstub in the debug process.

Internal Name:

```
debug.attach-defaults
```

Data Specification:

```
[tuple of: [tuple length 2 of: <type str>, [from 0 to 65535]]]
```

Default Value:

```
(('127.0.0.1', 50015),)
```

- **Advanced**

Network Timeout

Controls the amount of time that the debug client will wait for the debug server to respond before it gives up. This protects the IDE from freezing up if your program running within the debug server crashes (or if the server itself becomes unavailable). It must also be taken into account when network connections are slow or if sending large data values (see the huge-list-threshold and huge-string-threshold preferences).

Internal Name:

```
debug.network-timeout
```

Data Specification:

```
<type float>, <type int>
```

Default Value:

```
10
```

Show Data Warnings

Controls whether or not time out, huge value, and error handling value errors are displayed by the debugger the first time they are encountered in each run of Wing.

Internal Name:

```
debug.show-debug-data-warnings
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Use sys.stdin Wrapper

Whether sys.stdin should be set a wrapper object for user input in the program being debugged. The wrapper allows debug commands, such as pause, to be executed while the program is waiting for user input. The wrapper may cause problems with multi-threaded programs that use C stdio functions to read directly from stdin and will be slower than the normal file object.However, turning this preference off means that your debug process will not pause or accept breakpoint changes while waiting for keyboard input, and any keyboard input that occurs as a side effect of commands typed in the Debug Probe will happen in unmodified erb!stdin! instead (even though output will still appear in the Debug Probe as always).

Internal Name:

```
debug.use-stdin-wrapper
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Debug Internals Log File

This is used to obtain verbose information about debugger internals in cases where you are having problems getting debugging working. When set to non-None value, debugger activity is logged to the given file name. Alternatively, "<stdout>" or "<stderr>" can be used.

Internal Name:

```
debug.logfile
```

Data Specification:

```
[one of: None, [<stdout>, <stderr>], <type str>]
```

Default Value:

```
None
```

Shells Ignore Editor Modes

Set to False so that shells will act modal in the same way as editors when working with a modal key bindings such as that for VI. When True, the shells always act as if in Insert mode.

Internal Name:

```
debug.shells-ignore-editor-modes
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Source Analysis

Analyze in Background

Whether Wing should try to analyze python source in the background.

Internal Name:

```
pysource.analyze-in-background
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Max Cache Size (MB)

The maximum size of the disk cache in megabytes

Internal Name:

```
pysource.max-disk-cache-size
```

Data Specification:

```
[from 1 to 1000]
```

Default Value:

```
50
```

Max Memory Buffers

The maximum # of analysis info buffers that can be in-memory at once for files that are not open.

Internal Name:

```
pysource.max-background-buffers
```

Data Specification:

```
[from 1 to 100]
```

Default Value:

```
40
```

Typing Suspend Timeout

Number of seconds between last key press and when analysis is re-enabled if analyis is to be suspended while typing occurs. If <= 0, analysis is not suspended.

Internal Name:

```
edit.suspend-analysis-timeout
```

Data Specification:

```
<type float>, <type int>
```

Default Value:

```
3
```

- **Advanced**

Interface File Path

Path to search for interface files for extension modules. If directory name is relative, it will be interpreted as relative to the user settings directory (USER_SETTINGS_DIR)

Internal Name:

```
pysource.interfaces-path
```

Data Specification:

```
[tuple of: <type str>]
```

Default Value:

```
('pi-files',)
```

Scrape Extension Modules

Set this to False to disable automatic loading of extension modules and other modules that cannot be statically analysed. These modules are loaded in another process space and 'scraped' to obtain at least some analysis of the module's contents.

Internal Name:

```
pysource.scrape-modules
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
True
```

Scraping Helper Snippets

This is a dictionary from module name to Python code that should be executed before attempting to load extension modules for scraping. This is needed in some cases such as PyGTK and wxPython because the extension modules are designed to be loaded only after some configuration magic is performed. For most extension modules, no extra configuration should be needed.

Internal Name:

```
pysource.scrape-config
```

Data Specification:

```
[dict; keys: <type str>, values: <type str>]
```

Default Value:

```
{'QtSql': 'from PyQt4 import QtSql', 'QtGui': 'from PyQt4 im-
port QtGui', 'QtAssistant': 'from PyQt4 import QtAssis-
tant', 'gtk': 'import pygtk\nvers = pygtk._get_available_versions().keys()\n
cept:\n    pass\n', 'atk': 'import pygtk\nvers = pygtk._get_available_versio
cept:\n    pass\n', 'QtSvg': 'from PyQt4 import QtSvg', 'QtTest': 'from PyQt
port QtTest', 'wxpython': 'pass', 'QtOpenGL': 'from PyQt4 im-
port QtOpenGL', 'QtXml': 'from PyQt4 import QtXml', 'gob-
ject': 'import pygtk\nvers = pygtk._get_available_versions().keys()\nvers.so
cept:\n    pass\n', 'pango': 'import pygtk\nvers = pygtk._get_available_vers
cept:\n    pass\n', 'gdk': 'import pygtk\nvers = pygtk._get_available_versio
cept:\n    pass\n', 'QtCore': 'from PyQt4 import QtCore', 'Qt-
Network': 'from PyQt4 import QtNetwork'}
```

IDE Extension Scripting

Search Path

Specifies the directories in which Wing will look for user-defined scripts that extend the functionality of the IDE itself. For each directory, Wing will load all found Python modules and packages, treating any function whose name starts with a letter (not _ or __) as a script-provided command. Extension scripts found in files within directories later in the list will override scripts of the same name found earlier, except that scripts can never override commands that are defined internally in Wing itself (these are documented in the Command Reference in the users manual). See the Scripting and Extending chapter of the manual for more information on writing and using extension scripts. Note that WINGHOME/scripts is always appended to the given path since it contains scripts that ship with Wing.

Internal Name:

```
main.script-path
```

Data Specification:

```
[list of: <type str>]
```

Default Value:

```
[u'USER_SETTINGS_DIR/scripts']
```

Auto-Reload Scripts on Save

When enabled, Wing will automatically reload scripts that extend the IDE when they are edited and saved from the IDE. This makes developing extension scripts for the IDE very fast, and should work in most cases. Disable this when working on extension scripts that do not reload properly, such as those that reach through the scripting API extensively.

Internal Name:

```
main.auto-reload-scripts
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
True
```

Internal Preferences

Core Preferences

main.debug-break-on-critical

If True and a gtk, gdk, or glib critical message is logged, Wing tries to start a C debugger and break at the current execution point

Internal Name:

```
main.debug-break-on-critical
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
False
```

main.documentation-language

The language to use for the documentation, when available (not all documentation is translated into all supported languages).

Internal Name:

```
main.documentation-language
```

Data Specification:

```
[None, de, en, fr]
```

Default Value:

```
en
```

main.extra-mime-type-comments

This is a map from mime type to tuple of start/end comment characters for each mime type. One entry should be added for each new mime type added with the main.extra-mime-types preference.

Internal Name:

```
main.extra-mime-type-comments
```

Data Specification:

```
[dict; keys: <type str>, values: [tuple length 2 of: <type str>, <type str>]
```

Default Value:

```
{}
```

main.extra-mime-type-names

This is a map from mime type to displayable name for that mime type; one entry should be added for each new mime type added with the main.extra-mime-types preference.

Internal Name:

```
main.extra-mime-type-names
```

Data Specification:

```
[dict; keys: <type str>, values: <type str>]
```

Default Value:

```
{}
```

main.ignored-updates

Used internally to keep track of updates the user is not interested in

Internal Name:

```
main.ignored-updates
```

Data Specification:

```
[list of: <type str>]
```

Default Value:

```
[]
```

User Interface Preferences

gui.apple-keyboard

Whether an Apple keyboard is in use. Use query x11 option to attempt to determine setting from X11 server each time Wing is run. This is an OS X only preference.

Internal Name:

```
gui.apple-keyboard
```

Data Specification:

```
[query-x11, yes, no]
```

Default Value:

```
query-x11
```

gui.feedback-email

Email address to use by default in the Feedback and Bug Report dialogs

Internal Name:

```
gui.feedback-email
```

Data Specification:

```
<type str>
```

Default Value:

```
""
```

gui.fix-osx-tiger-keyboard-conflict

Whether to fix the inability to use Mode_switch on Tiger (OS X 10.4). If true, Wing will run xmodmap when it starts to remap the Mode_switch keys (option, Alt Gr, and other composition keys on non-US keyboards) from mod1 to mod5. The xmodmap modifications will affect all X11 applications.

Internal Name:

```
gui.fix-osx-tiger-keyboard-conflict
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
True
```

gui.osx-key-for-alt

Use key for alt key in all X11 applications on OS X -- typically used when using a non OS X keyboard layout on the Apple X11 server. This will use xmodmap to set the global X11 key map to use the specified key as the alt key modifier. Turning this option off if it was on previously will reset the option key back to mode_switch, which is the Apple default setting. Non-default options will override any externally set xmodmap setting so use with care if you've customized your xmodmap.

Internal Name:

```
gui.osx-key-for-alt
```

Data Specification:

```
[default, command, option]
```

Default Value:

```
default
```

gui.include-file-types

Controls which file types to include for multi-file operations such as searching and importing files into a project

Internal Name:

```
gui.include-file-types
```

Data Specification:

```
[tuple of: <type str>]
```

Default Value:

```
('*.*',)
```

gui.last-feedback-shown

Used internally to avoid showing the feedback dialog on exit over and over again.

Internal Name:

```
gui.last-feedback-shown
```

Data Specification:

```
<type float>
```

Default Value:

```
0.0
```

gui.omit-file-types

Lists file types that should be omitted from multi-file operations such as searching and importing files into a project. These are omitted even if the gui.include-file-types preference includes a matching wild card.

Internal Name:

```
gui.omit-file-types
```

Data Specification:

```
[tuple of: <type str>]
```

Default Value:

```
('*.o', '*.a', '*.so', '*.pyc', '*.pyo', 'core', '*~', '#*#', 'CVS', '.#*')
```

gui.prefered-symbol-order

Control preferred order in source index displays such as the editor browse menus. Either sort in "file-order" or "alpha-order".

Internal Name:

```
gui.prefered-symbol-order
```

Data Specification:

```
[file-order, alpha-order]
```

Default Value:

```
alpha-order
```

gui.reported-exceptions

Used internally to remember which unexpected exceptions have already been reported so we only show error reporting dialog once for each. This is a dict from product version to dict of exception info.

Internal Name:

```
gui.reported-exceptions
```

Data Specification:

```
[dict; keys: <type str>, values: [dict; keys: <type str>, val-
ues: <boolean: 0 or 1>]]
```

Default Value:

```
{}
```

gui.scan-for-pythoncom-shell-extensions

Scan for pythoncom shell extensions on Windows

Internal Name:

```
gui.scan-for-pythoncom-shell-extensions
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
True
```

gui.set-win32-foreground-lock-timeout

Controls whether or not to set the foreground lock timeout on Windows 98/ME and 2K/XP. On these systems, normally Wing will be unable to bring source windows to front whenever the debug process has windows in the foreground. When this preference is true, the system-wide value that prevents background applications from raising windows is cleared whenever Wing is running. This means that other apps will also be able to raise windows without these restrictions while Wing is running. Set the preference to false to avoid this, but be prepared for windows to fail to raise in some instances. Note: If Wing is terminated abnormally or from the task manager, the changed value will persist until the user logs out (or reboot on 98/ME).

Internal Name:

```
gui.set-win32-foreground-lock-timeout
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

gui.show-feedback-dialog

Whether feedback dialog is shown to user on quit.

Internal Name:

```
gui.show-feedback-dialog
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

gui.show-osx-keyboard-warning

Used internally to show information about osx keyboard issues to new users. Once turned off, it is never turned on again

Internal Name:

```
gui.show-osx-keyboard-warning
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

gui.startup-show-wingtips

Controls whether or not the Wing Tips tool is shown automatically at startup of the IDE.

Internal Name:

```
gui.startup-show-wingtips
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

Editor Preferences

edit.fold-mime-types

Set to a list of mime types for which folding should be allowed when folding in general is enabled.

Internal Name:

```
edit.fold-mime-types
```

Data Specification:

```
[list of: <type str>]
```

Default Value:

```
['text/x-python', 'text/x-c-source', 'text/x-cpp-source', 'text/x-
java-source', 'text/x-javascript', 'text/html', 'text/xml', 'text/x-
zope-pt', 'text/x-eiffel', 'text/x-lisp', 'text/x-ruby']
```

edit.use-default-foreground-when-printing

Use default foreground color for all text when printing. It's to set this if foreground color are customized for display on a dark background. The background color when printing is assumed to be white.

Internal Name:

```
edit.use-default-foreground-when-printing
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
False
```

Project Manager Preferences

proj.follow-selection

Controls whether or not the IDE will follow the current project manager selection by opening the corresponding source file in a non-sticky (auto-closing) editor. In either case, the project manager will always open a file in sticky mode when an item is double clicked or the Goto Source context menu item is used.

Internal Name:

```
proj.follow-selection
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Debugger Preferences

debug.auto-clear-debug-io

Set to automatically clear the debug I/O text each time a new debug session is started

Internal Name:

```
debug.auto-clear-debug-io
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

debug.default-python-exec

Sets the default Python Executable to use for debugging and source code analysis. This can be overridden on a project by project basis in Project Properties.

Internal Name:

```
debug.default-python-exec
```

Data Specification:

```
[None or <type str>]
```

Default Value:

```
None
```

debug.python-exec

Set this to override the default Python executable used with the debug server. A None (default) value uses /usr/bin/env python on Linux and the configured default on NT. Otherwise, give the full path of the python executable, e.g. /usr/local/bin/python or C:devpython. This preference only affects programs that are launched from the IDE.

Internal Name:

```
debug.python-exec
```

Data Specification:

```
[None or <type str>]
```

Default Value:

```
None
```

debug.safe-size-checks-only

This is a *temporary* preference that will go away in future version of Wing IDE. It can be used to turn off server-side size checking done on values typed in the interactive shell. When set to true, Wing may terminate the debug process on large values that are evaluated in the interactive shell. When set to false, Wing will do size checking to avoid such termination but will also cause duplicate execution of any functionality reached as the result of a __getattr__ method.

Internal Name:

```
debug.safe-size-checks-only
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

debug.shell-auto-restart-before-eval

Auto-restart the Python Shell before a file is evaluated within it. When this is disabled, be aware that previously defined symbols will linger in the Python Shell environment.

Internal Name:

```
debug.shell-auto-restart-before-eval
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

debug.shell-eval-whole-lines

Evaluate whole lines from editor rather than the exact selection, when a selection from the editor is sent to the Python Shell tool.

Internal Name:

```
debug.shell-eval-whole-lines
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

debug.shell-pasted-line-threshold

The number of lines after which the Python Shell will just print a summary rather than the actual lines of code pasted, dragged, or other transferred to the shell.

Internal Name:

```
debug.shell-pasted-line-threshold
```

Data Specification:

```
<type int>
```

Default Value:

```
10
```

debug.show-exceptions-tip

Used internally to show information about exception handling to new users. Once turned off, it is never turned on again

Internal Name:

```
debug.show-exceptions-tip
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

debug.stop-timeout

Number of seconds to wait before the debugger will stop in its own code after a pause request is received and no other Python code is reached.

Internal Name:

```
debug.stop-timeout
```

Data Specification:

```
<type int>, <type float>
```

Default Value:

```
3.0
```

debug.use-members-attrib

Set this to true to have the debug server use the __members__ attribute to try to interpret otherwise opaque data values. This is a preference because some extension modules contain bugs that result in crashing if this attribute is accessed. Note that __members__ has been deprecated since Python version 2.2.

Internal Name:

```
debug.use-members-attrib
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
1
```

debug.wrap-debug-io

Set to true to turn on line wrapping in the integrated debug I/O panel.

Internal Name:

```
debug.wrap-debug-io
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

debug.wrap-debug-probe

Set to true to turn on line wrapping in the integrated debug probe panel.

Internal Name:

```
debug.wrap-debug-probe
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

debug.wrap-python-shell

Set to true to turn on line wrapping in the Python shell panel.

Internal Name:

```
debug.wrap-python-shell
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Source Analysis Preferences

pysource.instance-attrib-scan-mode

How to scan for instance attributes.

Internal Name:

```
pysource.instance-attrib-scan-mode
```

Data Specification:

```
[init-only, all-methods]
```

Default Value:

```
all-methods
```

Source Browser Preferences

browser.follow-selection

Controls whether or not the IDE will follow the current browser selection by opening the corresponding source file in a non-sticky (auto-closing) editor. In either case, the browser will always open a file in sticky mode when an item is double clicked or the Goto Source context menu item is used.

Internal Name:

```
browser.follow-selection
```

Data Specification:

```
<boolean: 0 or 1>
```

Default Value:

```
0
```

Command Reference

This chapter describes the entire top-level command set of Wing IDE. Use this reference to look up command names for use in modified **keyboard bindings**.

Top Level Commands

These are the top-level application commands.

abandon-changes (confirm=True)

Abandon any changes in the current document and reload it from disk. Prompts for user to confirm the operation unless either there are no local changes being abandoned or confirm is set to False.

about-application ()

Show the application-wide about box

begin-visited-document-cycle (move_back=True)

Start moving between documents in the order they were visited. Starts modal key interation that ends when a key other than tab is seen or ctrl is released.

check-for-updates ()

Check for updates to Wing IDE and offer to install any that are available

close (ignore_changes=False, close_window=False)

Close active document. Abandon any changes when ignore_changes is True. Close empty windows and quit if all document windows closed when close_window is True.

close-all (omit_current=False, ignore_changes=False, close_window=False)

Close all documents in the current window, or in all windows if in one-window-per-editor

windowing policy. Leave currently visible documents (or active window in one-window-per-editor-mode) if omit_current is True. Abandons changes rather than saving them when ignore_changes is True. Close empty window and quit if all document windows closed when close_window is True.

close-window ()

Close the current window and all documents and panels in it

command-by-name (command_name)

Execute given command by name, collecting any args as needed

copy-tutorial ()

Prompt user and copy the tutorial directory from the Wing IDE installation to the directory selected by the user

edit-file-sets ()

Show the File Sets preference editor

edit-preferences-file ()

Edit the preferences as a text file

execute-cmd (cmd)

Execute the given command line silently in the background

execute-file (loc=None)

Execute the file at the given location or use the active view if loc is None.

goto-bookmark (mark)

Goto named bookmark

initiate-numeric-modifier (digit)

VI style repeat/numeric modifier for following command

initiate-repeat ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

initiate-repeat-0 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

initiate-repeat-1 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

initiate-repeat-2 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

initiate-repeat-3 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

initiate-repeat-4 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

initiate-repeat-5 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

initiate-repeat-6 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

initiate-repeat-7 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

initiate-repeat-8 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

initiate-repeat-9 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

internal-profile-start ()

Start internal profiling.

internal-profile-stop ()

Stop internal profiling.

new-blank-file (filename)

Create a new blank file on disk, open it in an editor, and add it to the current project.

new-document-window ()

Create a new document window with same documents and panels as in the current document window (if any; otherwise empty with default panels)

new-file (ext='.py')

Create a new file

new-panel-window (panel_type=None)

Create a new panel window of given type

next-document ()

Move to the next document alphabetically in the list of documents open in the current window

next-window ()

Switch to the next window alphabetically by title

open (filename)

Open a file from disk

open-gui (filename=None)

Open a file from disk, prompting with file selection dialog if necessary

previous-document ()

Move to the previous document alphabetically in the list of documents open in the current window

previous-window ()

None

query-end-session ()

Process query-end-session message on win32

quit ()

Quit the application.

recent-document ()

Switches to previous document most recently visited in the current window or window set if in one-window-per-editor windowing mode.

reload-scripts ()

Force reload of all scripts, from all configured script directories. This is usually only needed when adding a new script file. Existing scripts are automatically reloaded when they change on disk.

remove-bookmark (mark)

Remove the given named bookmark

restore-default-tools ()

Hide/remove all tools and restore to original default state

save (close=False, force=False)

Save active document. Also close it if close is True.

save-all (close_window=False)

Save all unsaved items. Will prompt the user only for choosing names for new items that don't have a set filename

save-as ()

Save active document to a new file

scratch-document (title='Scratch', mime_type='text/plain')

Create a new scratch buffer with given title and mime type. The buffer is never marked as changed but can be saved w/ save-as.

set-bookmark (mark)

Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

show-bookmarks ()

Show a list of all currently defined bookmarks

show-bug-report-dialog ()

Show the bug reporting dialog

show-document (section='manual')

Show the given documentation section

show-feedback-dialog ()

Show the feedback submission dialog

show-howtos ()

Show the How-Tos index

show-html-document (section='manual')

Show the given document section in HTML format.

show-manual-html ()

Show the HTML version of the Wing IDE users manual

show-manual-pdf ()

Show the PDF version of the Wing IDE users manual for either US Letter or A4, depending on user's print locale

show-panel (panel_type, flash=True)

Show most recently visited panel instance of given type If no such panel exists, add one to the primary window and show it. Returns the panel view object or None if not shown.

show-pdf-document (doc='manual')

Show the given document in PDF format. One of 'manual', 'intro', or 'howtos'.

show-preferences-gui (prefname=None)

Edit the preferences file using the preferences GUI, optionally opening to the section that contains the given preference by name

show-python-for-beginners-html ()

Show the Python for Beginners web page

show-python-introductions-html ()

Show the Python Introductions web page

show-python-manual-html ()

Show the HTML version of the Python users manual

show-python-org-html ()

Show the python.org site home page

show-python-org-search-html ()

Show the python.org site search page

show-quickstart ()

Show the quick start guide

show-success-stories-html ()

Show the Python Success Stories page

show-support-html ()

Show the Wing IDE support site home page

show-text-registers ()

Show the contents of all non-empty text registers in a temporary editor

show-tutorial ()

Show the tutorial

show-wingtip (section='/')

Show the Wing Tips window

switch-document (document_name)

Switches to named document. Name may either be the complete name or the last path component of a path name.

toolbar-search (text, next=False, set_anchor=True)

Search using text entered in toolbar search area from the current cursor or selection position

toolbar-search-next (text, set_anchor=True)

Move to next match of text entered in the toolbar search area

vi-goto-bookmark ()

Goto bookmark using single character name defined by the next pressed key

vi-set-bookmark ()

Set a bookmark at current location on the editor using the next key press as the name of the bookmark.

wing-tips ()

Display interactive tip manager

write-changed-file-and-close (filename)

Write current document to given location only if it contains any changes and close it. Writes to current file name if given filename is None.

write-file (filename)

Write current file to a new location.

write-file-and-close (filename)

Write current document to given location and close it. Saves to current file name if the given filename is None.

Dock Window Commands

Commands for windows that contain dockable tool areas. These are available for the currently active window, if any.

enter-fullscreen ()

Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen

exit-fullscreen ()

Restore previous non-fullscreen state of all tools and tool bar

hide-horizontal-tools ()

Hide the horizontal tool area

hide-vertical-tools ()

Hide the vertical tool area

minimize-horizontal-tools ()

Minimize the horizontal tool area

minimize-vertical-tools ()

Minimize the vertical tool area

show-horizontal-tools ()

Show the horizontal tool area

show-vertical-tools ()

Show the vertical tool area

toggle-horizontal-tools ()

Show or minimize the horizontal tool area

toggle-vertical-tools ()

Show or minimize the vertical tool area

Document Viewer Commands

Commands for the documentation viewer. These are available when the documentation viewer has the keyboard focus.

document-back ()

Go back to prior document page in history of pages that have been viewed

document-contents ()

Go to the document contents page

document-forward ()

Go forward to next document page in history of pages that have been viewed

document-next ()

Go to the next page in the current document

document-previous ()

Go to the previous page in the current document

isearch-backward (search_string=None, repeat=<command.commandmgr.kArgNumericModifie
instance at 0x4145f7cc>)

Initiate incremental mini-search backward from the cursor position, optionally entering the given search string.

isearch-backward-regex (search_string=None, repeat=<command.commandmgr.kArgNumeric
instance at 0x4145f7ec>)

Initiate incremental regular expression mini-search backward from the cursor position, optionally entering the given search string.

isearch-forward (search_string=None, repeat=<command.commandmgr.kArgNumericModifier
instance at 0x4145f6cc>)

Initiate incremental mini-search forward from the cursor position, optionally entering the given search string.

isearch-forward-regex (search_string=None, repeat=<command.commandmgr.kArgNumericM
instance at 0x4145f7ac>)

Initiate incremental regular expression mini-search forward from the cursor position, optionally entering the given search string.

isearch-repeat (reverse=False, repeat=<command.commandmgr.kArgNumericModifier
instance at 0x4145f80c>)

Repeat the most recent isearch, using same string and regex/text. Reverse direction when reverse is True.

isearch-sel-backward (persist=True, repeat=<command.commandmgr.kArgNumericModifier
instance at 0x4145f88c>)

Initiate incremental mini-search backward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

isearch-sel-forward (persist=True, repeat=<command.commandmgr.kArgNumericModifier
instance at 0x4145f84c>)

Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

repeat-search-char (opposite=0, repeat=<command.commandmgr.kArgNumericModifier instance at 0x4145f8cc>)

Repeat the last search_char operation, optionally in the opposite direction.

search-char (dir=1, pos=0, repeat=<command.commandmgr.kArgNumericModifier instance at 0x4145f8ac>, single_line=0)

Search for the given character. Searches to right if dir > 0 and to left if dir < 0. Optionally place cursor pos characters to left or right of the target (e.g., use -1 to place one to left). If repeat > 1, the Nth match is found. Set single_line=1 to search only within the current line.

Editor Browse Mode Commands

Commands available only when the editor is in browse mode (used for VI bindings and possibly others)

enter-insert-mode (pos='before')

Enter editor insert mode

enter-replace-mode ()

Enter editor replace mode

enter-visual-mode (unit='char')

Enter editor visual mode. Unit should be one of 'char', 'line', or 'block'.

start-select-block ()

Turn on auto-select block mode

start-select-char ()

Turn on auto-select mode character by character

start-select-line ()

Turn on auto-select mode line by line

vi-command-by-name ()

Execute a VI command (implements ":" commands from VI)

vi-ctrl-c ()

Perform vi mode ctrl-c action which either does a copy or nothing if ctrl-x/v/c are not being used for clipboard actions. The default is to map ctrl-c to clipboard on Windows and OS X. This can be overridden by the VI Mode Ctrl-X/C/V preference.

vi-ctrl-v ()

Perform vi mode ctrl-v action which either does a paste or does start-select-block. The default is to map ctrl-v to clipboard on Windows and OS X. This can be overridden by the VI Mode Ctrl-X/C/V preference.

vi-ctrl-x ()

Perform vi mode ctrl-x action which either does a cut or does initiate-numeric-modified with the following digit key press. The default is to map ctrl-x to clipboard on Windows and OS X. This can be overridden by the VI Mode Ctrl-X/C/V preference.

Editor Insert Mode Commands

Commands available only when editor is in insert mode (used for VI bindings and possibly others)

enter-browse-mode (provisional=False)

Enter editor browse mode

vi-ctrl-c ()

Perform vi mode ctrl-c action which either does a copy or enters browse mode if ctrl-x/v/c are not being used for clipboard actions. The default is to map ctrl-c to clipboard on Windows and OS X. This can be overridden by the VI Mode Ctrl-X/C/V preference.

vi-ctrl-v ()

Perform vi mode ctrl-v action which either does a paste or does start-select-block. The default is to map ctrl-v to clipboard on Windows and OS X. This can be overridden by the VI Mode Ctrl-X/C/V preference.

vi-ctrl-x ()

Perform vi mode ctrl-x action which either does a cut or nothing depending on whether ctrl-x/v/c are mapped to clipboard actions. The default is to map ctrl-x to clipboard on Windows and OS X. This can be overridden by the VI Mode Ctrl-X/C/V preference.

Editor Non Modal Commands

Commands available only when the editor is in non-modal editing mode

exit-visual-mode ()

None

start-select-block ()

Turn on auto-select block mode

start-select-char ()

Turn on auto-select mode character by character

start-select-line ()

Turn on auto-select mode line by line

Editor Panel Commands

Commands that control splitting up an editor panel. These are available when one split in the editor panel has the keyboard focus.

split-horizontally (new=0)

Split current view horizontally.

split-horizontally-open-file (filename)

Split current view horizontally and open selected file

split-vertically (new=0)

Split current view vertically. Create new editor in new view when new==1.

split-vertically-open-file (filename)

Split current view vertically and open selected file

unsplit (action='current')

Unsplit all editors so there's only one. Action specifies how to choose the remaining displayed editor. One of:

current -- Show current editor close -- Close current editor before unsplitting recent -- Change to recent buffer before unsplitting recent-or-close -- Change to recent buffer before closing

Unexpected indentation.

split, or close the current buffer if there is only one split left.

NOTE: The parameters for this command are subject to change in the future.

Editor Replace Mode Commands

Commands available only when editor is in replace mode (used for VI bindings and possibly others)

enter-browse-mode (provisional=False)

Enter editor browse mode

Editor Split Commands

Commands for a particular editor split, available when the editor in that split has the keyboard focus. Additional commands affecting the editor's content are defined separately.

activate-file-option-menu ()

Activate the file menu for the editor.

grow-split-horizontally ()

Increase width of this split

grow-split-vertically ()

Increase height of this split

next-bookmark ()

Move forward to next auto-bookmarked editor position

previous-bookmark ()

Move back to previous auto-bookmarked editor position

shrink-split-horizontally ()

Decrease width of this split

shrink-split-vertically ()

Decrease height of this split

Editor Visual Mode Commands

Commands available only when the editor is in visual mode (used for VI bindings and some others)

enter-browse-mode ()

Enter editor browse mode

enter-insert-mode (pos='delete-sel')

Enter editor insert mode

enter-visual-mode (unit='char')

Alter type of editor visual mode or exit back to browse mode. Unit should be one of 'char', 'line', or 'block'.

exit-visual-mode ()

Exit visual mode and return back to default mode

vi-command-by-name ()

Execute a VI command (implements ":" commands from VI)

Global Documentation Commands

Commands for the documentation viewer. These are available when the documentation viewer has the keyboard focus.

document-search (txt=None)

Search all documentation.

Toolbar Search Commands

Commands available when the tool bar search entry area has the keyboard focus.

backward-char ()

Move backward one character

backward-char-extend ()

Move backward one character, extending the selection

backward-delete-char ()

Delete character behind the cursor

backward-delete-word ()

Delete word behind the cursor

backward-word ()

Move backward one word

backward-word-extend ()

Move backward one word, extending the selection

beginning-of-line ()

Move to the beginning of the toolbar search entry

beginning-of-line-extend ()

Move to the beginning of the toolbar search entry, extending the selection

copy ()

Cut selection

cut ()

Cut selection

end-of-line ()

Move to the end of the toolbar search entry

end-of-line-extend ()

Move to the end of the toolbar search entry, extendning the selection

forward-char ()

Move forward one character

forward-char-extend ()

Move forward one character, extending the selection

forward-delete-char ()

Delete character in front of the cursor

forward-delete-word ()

Delete word in front of the cursor

forward-word ()

Move forward one word

forward-word-extend ()

Move forward one word, extending the selection

paste ()

Paste from clipboard

Window Commands

Commands for windows in general. These are available for the currently active window, if any.

move-editor-focus (dir=1, wrap=True)

Move focus to next or previous editor split, optionally wrapping when the end is reached.

move-editor-focus-first ()

Move focus to first editor split

move-editor-focus-last ()

Move focus to last editor split

move-editor-focus-previous ()

Move focus to last editor split

move-focus ()

None

Wing Tips Commands

Commands for the Wing Tips tool. These are only available when the tool is visible and has focus

wingtips-close ()

Close the Wing Tips window

wingtips-contents ()

Got to the Wing Tips contents page

wingtips-next ()

Got to the next page in Wing Tips

wingtips-next-unseen ()

Got to a next unseen Wing Tips page

wingtips-previous ()

Got to the previous page in Wing Tips

Active Editor Commands

Commands that only apply to editors when they have the keyboard focus. These commands are also available for the Python Shell, Debug Probe, and Debug I/O tools, which subclass the source editor, although some of the commands are modified or disabled as appropriate in those contexts.

activate-symbol-option-menu-1 ()

Activate the 1st symbol menu for the editor.

activate-symbol-option-menu-2 ()

Activate the 2nd symbol menu for the editor.

activate-symbol-option-menu-3 ()

Activate the 3rd symbol menu for the editor.

activate-symbol-option-menu-4 ()

Activate the 4th symbol menu for the editor.

backward-char (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1e0c>)

Move cursor backward one character

backward-char-extend (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1e4c>)

Move cursor backward one character, adjusting the selection range to new position

backward-char-extend-rect (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1e8c>)

Move cursor backward one character, adjusting the rectangular selection range to new position

backward-delete-char (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb2ac>)

Delete one character behind the cursor, or the current selection if not empty.

backward-delete-word (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb2ec>)

Delete one word behind of the cursor

backward-page (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb0ec>)

Move cursor backward one page

backward-page-extend (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb12c>)

Move cursor backward one page, adjusting the selection range to new position

backward-paragraph (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1fcc>)

Move cursor backward one paragraph (to next all-whitespace line).

backward-paragraph-extend (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb02c>)

Move cursor backward one paragraph (to next all-whitespace line), adjusting the selection range to new position.

backward-tab ()

Outdent line at current position

backward-word (delimiters=None, gravity='start', repeat=<command.commandmgr.kArgNum instance at 0x412b1f0c>)

Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.

backward-word-extend (delimiters=None, gravity='start', repeat=<command.commandmgr.k instance at 0x412b1f2c>)

Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.

beginning-of-line ()

Move to beginning of current line, or to the end of the leading white space if already at the beginning of the line.

beginning-of-line-extend ()

Move to beginning of current line, or to the end of the leading white space if already at the beginning of the line, adjusting the selection range to the new position.

beginning-of-line-text ()

Move to end of the leading white space, if any, on the current line, or to the beginning of the line if already at the end of the leading white space.

beginning-of-line-text-extend ()

Move to end of the leading white space, if any, on the current line, or to the beginning of the line if already at the end of the leading white space. The selection range is adjusted to the new position.

beginning-of-screen-line ()

Move to beginning of current wrapped line

beginning-of-screen-line-extend ()

Move to beginning of current wrapped line, extending selection

beginning-of-screen-line-text ()

Move to first non-blank character at beginning of current wrapped line

beginning-of-screen-line-text-extend ()

Move to first non-blank character at beginning of current wrapped line, extending selection

brace-match ()

Match brace at current cursor position, selecting all text between the two and hilighting the braces

cancel ()

Cancel current editor command

cancel-autocompletion ()

Cancel any active autocompletion.

case-lower (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb56c>)

Change case of the current selection, or character ahead of the cursor if there is no selection, to lower case

case-lower-next-move (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb66c>)

Change case of text spanned by next cursor movement to lower case

case-swap (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb5ec>)

Change case of the current selection, or character ahead of the cursor if there is no selection, so each letter is the opposite of its current case

case-swap-next-move (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb6ec>)

Change case of text spanned by next cursor movement so each letter is the opposite of its current case

case-title (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb5ac>)

Change case of the current selection, or character ahead of the cursor if there is no selection, to title case (first letter of each word capitalized)

case-title-next-move (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb6ac>)

Change case of text spanned by next cursor movement to title case (first letter of each word capitalized)

case-upper (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb50c>)

Change case of the current selection, or character ahead of the cursor if there is no selection, to upper case

case-upper-next-move (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb62c>)

Change case of text spanned by next cursor movement to upper case

center-cursor ()

Scroll so cursor is centered on display

clear ()

Clear selected text

complete-autocompletion (append=")

Complete the current active autocompletion.

copy ()

Copy selected text

cursor-move-to-bottom (offset=<command.commandmgr.kArgNumericModifier instance at 0x412bb8ec>)

Move cursor to bottom of display (without scrolling), optionally at an offset of given number of lines before bottom

cursor-move-to-center ()

Move cursor to center of display (without scrolling)

cursor-move-to-top (offset=<command.commandmgr.kArgNumericModifier instance at 0x412bb8ac>)

Move cursor to top of display (without scrolling), optionally at an offset of given number of lines below top

cursor-to-bottom ()

Scroll so cursor is centered at bottom of display

cursor-to-top ()

Scroll so cursor is centered at top of display

cut ()

Cut selected text

cut-line ()

Cut the current line(s) to clipboard.

delete-line (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb32c>)

Delete the current line or lines when the selection spans multiple lines or given repeat is > 1

delete-line-insert (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb36c>)

Delete the current line or lines when the selection spans multiple lines or given repeat is > 1. Enters insert mode (when working with modal key bindings).

delete-next-move (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb44c>)

Delete the text covered by the next cursor move command.

delete-next-move-insert (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb48c>)

Delete the text covered by the next cursor move command and then enter insert mode (when working in a modal editor key binding)

delete-range (start_line, end_line, register=None)

Delete given range of lines, copying them into given register (or currently selected default register if register is None

delete-to-end-of-line (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb3ac>, post_offset=0)

Delete everything between the cursor and end of line

delete-to-end-of-line-insert (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb3cc>)

Delete everything between the cursor and end of line and enter insert move (when working in a modal editor key binding)

delete-to-start-of-line ()

Delete everything between the cursor and start of line

end-of-document ()

Move cursor to end of document

end-of-document-extend ()

Move cursor to end of document, adjusting the selection range to new position

end-of-line (count=<command.commandmgr.kArgNumericModifier instance at 0x412b1b0c>)

Move to end of current line

end-of-line-extend (count=<command.commandmgr.kArgNumericModifier instance at 0x412b1b2c>)

Move to end of current line, adjusting the selection range to new position

end-of-screen-line (count=<command.commandmgr.kArgNumericModifier instance at 0x412b1b4c>)

Move to end of current wrapped line

end-of-screen-line-extend (count=<command.commandmgr.kArgNumericModifier instance at 0x412b1b6c>)

Move to end of current wrapped line, extending selection

exchange-point-and-mark ()

When currently marking text, this exchanges the current position and mark ends of the current selection

filter-next-move (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb4cc>)

Filter the lines covered by the next cursor move command through an external command and replace the lines with the result

filter-range (cmd, start_line=0, end_line=-1)

Filter a range of lines in the editor through an external command and replace the lines with the result. Filters the whole file by default.

filter-selection (cmd)

Filter the current selection through an external command and replace the lines with the result

form-feed ()

Place a form feed character at the current cursor position

forward-char (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1d6c>)

Move cursor forward one character

forward-char-extend (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1dac>)

Move cursor forward one character, adjusting the selection range to new position

forward-char-extend-rect (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1dcc>)

Move cursor forward one character, adjusting the rectangular selection range to new position

forward-delete-char (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb1ac>)

Delete one character in front of the cursor

forward-delete-char-insert (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb1ec>)

Delete one char in front of the cursor and enter insert mode (when working in modal key bindings)

forward-delete-word (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb22c>)

Delete one word in front of the cursor

forward-delete-word-insert (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb26c>)

Delete one word in front of the cursor and enter insert mode (when working in modal key bindings)

forward-page (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb06c>)

Move cursor forward one page

forward-page-extend (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb0ac>)

Move cursor forward one page, adjusting the selection range to new position

forward-paragraph (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1f4c>)

Move cursor forward one paragraph (to next all-whitespace line).

forward-paragraph-extend (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1f8c>)

Move cursor forward one paragraph (to next all-whitespace line), adjusting the selection range to new position.

forward-tab ()

Place a tab character at the current cursor position

forward-word (delimiters=None, gravity='start', repeat=<command.commandmgr.kArgNumeri instance at 0x412b1ecc>)

Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.

forward-word-extend (delimiters=None, gravity='start', repeat=<command.commandmgr.kAr instance at 0x412b1eec>)

Move cursor forward one word, adjusting the selection range to new position. Optionally, rovide a string that contains the delimiters to define which characters are part of a word.

Gravity may be “start” or “end” to indicate whether cursor is placed at start or end of the word.

hide-selection ()

Turn off display of the current text selection

indent-to-match ()

Indent the current line or selected region to match indentation of preceding non-blank line

indent-to-next-indent-stop ()

Indent to next indent stop from the current position. Acts like indent command if selection covers multiple lines.

isearch-backward (search_string=None, repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b18ec>)

Initiate incremental mini-search backward from the cursor position, optionally entering the given search string

isearch-backward-regex (search_string=None, repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b192c>)

Initiate incremental regular expression mini-search backward from the cursor position, optionally entering the given search string

isearch-forward (search_string=None, repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b182c>)

Initiate incremental mini-search forward from the cursor position, optionally entering the given search string

isearch-forward-regex (search_string=None, repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b18cc>)

Initiate incremental regular expression mini-search forward from the cursor position, optionally entering the given search string

isearch-repeat (reverse=False, repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b196c>)

Repeat the most recent isearch, using same string and regex/text. Reverse direction when reverse is True.

isearch-sel-backward (persist=True, whole_word=False, repeat=<command.commandmgr.kArg
instance at 0x412b1a2c>)

Initiate incremental mini-search backward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

isearch-sel-forward (persist=True, whole_word=False, repeat=<command.commandmgr.kArgN
instance at 0x412b19ec>)

Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

kill-line ()

Kill rest of line from cursor to end of line, and place it into the clipboard with any other contiguously removed lines. End-of-line is removed only if there is nothing between the cursor and the end of the line.

middle-of-screen-line ()

Move to middle of current wrapped line

middle-of-screen-line-extend ()

Move to middle of current wrapped line, extending selection

move-to-register (unit='char', cut=0, num=<command.commandmgr.kArgNumericModifier
instance at 0x412a8f6c>)

Cut or copy a specified number of characters or lines, or the current selection. Set cut=1 to remove the range of text from the editor after moving to register (otherwise it is just copied). Unit should be one of 'char' or 'line' or 'sel' for current selection.

move-to-register-next-move (cut=0, repeat=<command.commandmgr.kArgNumericModifier
instance at 0x412b104c>)

Move the text spanned by the next cursor motion to a register

new-line ()

Place a new line at the current cursor position

next-line (cursor='same', repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1b8c>)

Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

next-line-extend (cursor='same', repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1bac>)

Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

next-line-extend-rect (cursor='same', repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1bec>)

Move to next screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

next-line-in-file (cursor='start', repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1cec>)

Move to next line in file, repositioning character within line: 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

paste ()

Paste text from clipboard

paste-register (pos=1, indent=0, cursor=-1)

Paste text from register as before or after the current position. If the register contains only lines, then the lines are pasted before or after current line (rather than at cursor). If the register contains fragments of lines, the text is pasted over the current selection or either before or after the cursor. Set pos = 1 to paste after, or -1 to paste before. Set indent=1 to indent the pasted text to match current line. Set cursor=-1 to place cursor before lines or cursor=1 to place it after lines after paste completes.

previous-line (cursor='same', repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1c2c>)

Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

previous-line-extend (cursor='same', repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1c6c>)

Move to previous screen line, adjusting the selection range to new position, optionally

repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

previous-line-extend-rect (cursor='same', repeat=<command.commandmgr.kArgNumericMod instance at 0x412b1cac>)

Move to previous screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

previous-line-in-file (cursor='start', repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1d2c>)

Move to previous line in file, repositioning character within line: 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

profile-editor-start ()

Turn on profiling for the current source editor

profile-editor-stop ()

Stop profiling and print stats to stdout

reanalyze-file ()

Rescan file for code analysis.

redo ()

Redo last action

repeat-command (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1acc>)

Repeat the last editor command

repeat-search-char (opposite=0, repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1a6c>)

Repeat the last search_char operation, optionally in the opposite direction.

rstrip-each-line ()

Strip trailing whitespace from each line.

scroll-text-down (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb7ec>)

Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen.

scroll-text-left (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb82c>)

Scroll text left a column w/o moving cursor's relative position on screen. Repeat is number of columns or if >0 and <1.0 then percent of screen.

scroll-text-page-down (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb76c>)

Scroll text down a page w/o moving cursor's relative position on screen. Repeat is number of pages or if >0 and <1.0 then percent of screen.

scroll-text-page-up (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb72c>)

Scroll text up a page w/o moving cursor's relative position on screen. Repeat is number of pages or if >0 and <1.0 then percent of screen.

scroll-text-right (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb86c>)

Scroll text right a column w/o moving cursor's relative position on screen. Repeat is number of columns or if >0 and <1.0 then percent of screen.

scroll-text-up (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb7ac>)

Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen.

scroll-to-cursor ()

Scroll to current cursor position, if not already visible

search-char (dir=1, pos=0, repeat=<command.commandmgr.kArgNumericModifier instance at 0x412b1a4c>, single_line=0)

Search for the given character. Searches to right if dir > 0 and to left if dir < 0. Optionally place cursor pos characters to left or right of the target (e.g., use -1 to place one to left). If repeat > 1, the Nth match is found. Set single_line=1 to search only within the current line.

select-all ()

Select all text in the editor

set-mark-command (unit='char')

Set start of text marking for selection at current cursor position. Subsequently, all cursor move operations will automatically extend the text selection until stop-mark-command is issued. Unit defines what is selected: can be one of char, line, or block (rectangle).

set-register ()

Set the register to use for subsequent cut/copy/paste operations

show-autocompleter ()

Show the auto-completer for current cursor position

show-selection ()

Turn on display of the current text selection

start-of-document ()

Move cursor to start of document

start-of-document-extend ()

Move cursor to start of document, adjusting the selection range to new position

stop-mark-command (deselect=True)

Stop text marking for selection at current cursor position, leaving the selection set as is. Subsequent cursor move operations will deselect the range and set selection to cursor position. Deselect immediately when deselect is True.

tab-key ()

Implement the tab key, the action of which is configurable by preference

undo ()

Undo last action

yank-line ()

Yank contents of kill buffer created with kill-line into the edit buffer

General Editor Commands

Editor commands that act on the current (most recently active) source editor, whether or not it currently has the keyboard focus.

check-indent-consistency ()

Check whether indents consistently use spaces or tabs throughout the file.

comment-out-region ()

Comment out the selected region

convert-indents-to-mixed ()

Convert all lines with leading spaces to mixed tabs and spaces.

convert-indents-to-spaces-only ()

Convert all lines containing leading tabs to spaces only.

convert-indents-to-tabs-only ()

Convert all indentation to use tab characters only and no spaces

evaluate-file-in-shell (restart_shell=None)

Run the contents of the editor within the Python Shell

evaluate-sel-in-shell (restart_shell=False, whole_lines=None)

Evaluate the current selection from the editor within the Python Shell tool, optionally restarting the shell first. When whole_lines is set, the selection is rounded to whole lines before evaluation. When unspecified (set to None), the setting from the Shell's Option menu is used instead.

execute-kbd-macro (register='a', repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bbb2c>)

Execute most recently recorded keyboard macro. If register is None then the user is asked to enter a letter a-z for the register where the macro is filed. Otherwise, register 'a' is used by default.

fill-paragraph ()

Attempt to auto-justify the paragraph around the current start of selection

fold-collapse-all ()

Collapse all fold points in the current file

fold-collapse-all-clicked ()

Collapse the clicked fold point completely

fold-collapse-all-current ()

Collapse the current fold point completely

fold-collapse-more-clicked ()

Collapse the clicked fold point one more level

fold-collapse-more-current ()

Collapse the current fold point one more level

fold-expand-all ()

Expand all fold points in the current file

fold-expand-all-clicked ()

Expand the clicked fold point completely

fold-expand-all-current ()

Expand the current fold point completely

fold-expand-more-clicked ()

Expand the clicked fold point one more level

fold-expand-more-current ()

Expand the current fold point one more level

fold-toggle ()

Toggle the current fold point

fold-toggle-clicked ()

Toggle the clicked fold point

force-indent-style-to-match-file ()

Force the indent style of the editor to match the indent style found in the majority of the file

force-indent-style-to-mixed ()

Force the indent style of the editor to mixed use of tabs and spaces, regardless of the file contents

force-indent-style-to-spaces-only ()

Force the indent style of the editor to use spaces only, regardless of file contents

force-indent-style-to-tabs-only ()

Force the indent style of the editor to use tabs only, regardless of file contents

goto-clicked-symbol-defn ()

Goto the definition of the source symbol that was last clicked on

goto-column (column=<command.commandmgr.kArgNumericModifier instance at 0x412bbacc>)

Move cursor to given column

goto-line (lineno=<command.commandmgr.kArgNumericModifier instance at 0x412bb9cc>)

Position cursor at start of given line number

goto-nth-line (lineno=<command.commandmgr.kArgNumericModifier instance at 0x412bba0c>, cursor='start')

Position cursor at start of given line number (1=first, -1 = last). This differs from goto-line in that it never prompts for a line number but instead uses the previously entered numeric modifier or defaults to going to line one. The cursor can be positioned at 'start', 'end', or 'fnb' for first non-blank character.

goto-nth-line-default-end (lineno=<command.commandmgr.kArgNumericModifier instance at 0x412bba4c>, cursor='start')

Same as goto_nth_line but defaults to end of file if no lineno is given

goto-percent-line (percent=<command.commandmgr.kArgNumericModifier instance at 0x412bba8c>, cursor='start')

Position cursor at start of line at given percent in file. This uses the previously entered numeric modifier or defaults to going to line one. The cursor can be positioned at 'start', 'end', or 'fnb' for first non-blank character, or in VI mode it will do brace matching operation to reflect how VI overrides this command.

goto-selected-symbol-defn ()

Goto the definition of the selected source symbol

hide-all-whitespace ()

Turn off all special marks for displaying white space and end-of-line

hide-eol ()

Turn off special marks for displaying end-of-line chars

hide-indent-guides ()

Turn off special marks for displaying indent level

hide-whitespace ()

Turn off special marks for displaying white space

indent-lines (num=<command.commandmgr.kArgNumericModifier instance at 0x412bbbac>)

Indent selected number of lines from cursor position

indent-next-move (num=<command.commandmgr.kArgNumericModifier instance at 0x412bbc4c>)

Indent lines spanned by next cursor move

indent-region (sel=None)

Indent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

indent-to-match-next-move (num=<command.commandmgr.kArgNumericModifier instance at 0x412bbccc>)

Indent lines spanned by next cursor move to match, based on the preceding line

insert-command (cmd)

Insert the output for the given command at current cursor position. Some special characters in the command line (if not escaped with) will be replaced as follows:

Unexpected indentation.

% -- Current file's full path name # -- Previous file's full path name

insert-file (filename)

Insert a file at current cursor position, prompting user for file selection

join-lines (delim=' ', num=<command.commandmgr.kArgNumericModifier instance at 0x412bbd6c>)

Join together specified number of lines after current line (replace newlines with the given delimiter (single space by default)

join-selection (delim=' ')

Join together all lines in given selection (replace newlines with the given delimiter (single space by default)

kill-buffer ()

Close the current text file

outdent-lines (num=<command.commandmgr.kArgNumericModifier instance at 0x412bbc0c>)

Outdent selected number of lines from cursor position

outdent-next-move (num=<command.commandmgr.kArgNumericModifier instance at 0x412bbc8c>)

Outdent lines spanned by next cursor move

outdent-region (sel=None)

Outdent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

page-setup ()

Show printing page setup dialog

print-view ()

Print active editor document

query-replace (search_string, replace_string)

Initiate incremental mini-search query/replace from the cursor position.

query-replace-regex (search_string, replace_string)

Initiate incremental mini-search query/replace from the cursor position. The search string is treated as a regular expression.

range-replace (search_string, replace_string, confirm, range_limit, match_limit, regex)

Initiate incremental mini-search query/replace within the given selection. This is similar to query_replace but allows some additional options: confirm -- True to confirm each replace range_limit -- None to replace between current selection start and end of document,

Unexpected indentation.

> 1 to limit operation to current selection or to current line of selection is empty, (start, end) to limit operation to within given selection range, or "first|last" to limit operating withing given range of lines.

Block quote ends without a blank line; unexpected unindent.

match_limit -- None to replace any number of matches, or limit of number of replaces regex -- Treat search string as a regular expression

repeat-replace (repeat=<command.commandmgr.kArgNumericModifier instance at 0x412bb96c>)

Repeat the last query replace or range replace operation on the current line. The first match is replaced without confirmation.

replace-char (line_mode='multiline', num=<command.commandmgr.kArgNumericModifier instance at 0x412bb9ac>)

Replace num characters with given character. Set line_mode to multiline to allow replacing across lines, extend to replace on current line and then extend the line length, and restrict to replace only if enough characters exist on current line after cursor position.

replace-string (search_string, replace_string)

Replace all occurrences of a string from the cursor position to end of file.

replace-string-regex (search_string, replace_string)

Replace all occurrences of a string from the cursor position to end of file. The search string is treated as a regular expression.

save-buffer ()

Save the current text file to disk

set-readonly ()

Set editor to be readonly. This cannot be done if the editor contains any unsaved edits.

set-writable ()

Set editor to be writable. This can be used to override the read-only state used initially for editors displaying files that are read-only on disk.

show-all-whitespace ()

Turn on all special marks for displaying white space and end-of-line

show-eol ()

Turn on special marks for displaying end-of-line chars

show-indent-guides ()

Turn on special marks for displaying indent level

show-indent-manager ()

Display the indentation manager for this editor file

show-whitespace ()

Turn on special marks for displaying white space

start-kbd-macro (register='a')

Start definition of a keyboard macro. If register=None then the user is prompted to enter a letter a-z under which to file the macro. Otherwise, register 'a' is used by default.

stop-kbd-macro ()

Stop definition of a keyboard macro

toggle-line-wrapping ()

Toggles line wrapping preference for all editors

toggle-overtype ()

Toggle status of overtyping mode

uncomment-out-region ()

Uncomment out the selected region

use-lexer-ada ()

Force syntax highlighting Ada source

use-lexer-apache-conf ()

Force syntax highlighting for Apache configuration file format

use-lexer-asm ()

Force syntax highlighting for Masm assembly language

use-lexer-ave ()

Force syntax highlighting for Avenue GIS language

use-lexer-baan ()

Force syntax highlighting for Baan

use-lexer-bash ()

Force syntax highlighting for bash scripts

use-lexer-bullant ()

Force syntax highlighting for Bullant

use-lexer-by-doctype ()

Use syntax highlighting appropriate to the file type

use-lexer-cpp ()

Force syntax highlighting for C/C++ source

use-lexer-css2 ()

Force syntax highlighting for CSS2

use-lexer-diff ()

Force syntax highlighting for diff/cdiff files

use-lexer-dos-batch ()

Force syntax highlighting for DOS batch files

use-lexer-eiffel ()

Force syntax highlighting for Eiffel source

use-lexer-errlist ()

Force syntax highlighting for error list format

use-lexer-escript ()

Force syntax highlighting for EScript

use-lexer-fortran ()

Force syntax highlighting for Fortran

use-lexer-html ()

Force syntax highlighting for HTML

use-lexer-idl ()

Force syntax highlighting for XP IDL

use-lexer-java ()

Force syntax highlighting for Java source

use-lexer-javascript ()

Force syntax highlighting for Javascript

use-lexer-latex ()

Force syntax highlighting for LaTeX

use-lexer-lisp ()

Force syntax highlighting for Lisp source

use-lexer-lout ()

Force syntax highlighting for LOUT typesetting language

use-lexer-lua ()

Force syntax highlighting for Lua

use-lexer-makefile ()

Force syntax highlighting for make files

use-lexer-matlab ()

Force syntax highlighting for Matlab

use-lexer-mmixal ()

Force syntax highlighting for MMIX assembly language

use-lexer-msidl ()

Force syntax highlighting for MS IDL

use-lexer-nncrontab ()

Force syntax highlighting for NNCrontab files

use-lexer-none ()

Use no syntax highlighting

use-lexer-nsis ()

Force syntax highlighting for NSIS

use-lexer-pascal ()

Force syntax highlighting for Pascal source

use-lexer-perl ()

Force syntax highlighting for Perl source

use-lexer-php ()

Force syntax highlighting for PHP source

use-lexer-plsql ()

Force syntax highlighting for PL/SQL files

use-lexer-pov ()

Force syntax highlighting for POV ray tracer scene description language

use-lexer-properties ()

Force syntax highlighting for properties files

use-lexer-ps ()

Force syntax highlighting for Postscript

use-lexer-python ()

Force syntax highlighting for Python source

use-lexer-rc ()

Force syntax highlighting for RC file format

use-lexer-ruby ()

Force syntax highlighting for Ruby source

use-lexer-scriptol ()

Force syntax highlighting for Scriptol

use-lexer-sql ()

Force syntax highlighting for SQL

use-lexer-tcl ()

Force syntax highlighting for TCL

use-lexer-vb ()

Force syntax highlighting for Visual Basic

use-lexer-vxml ()

Force syntax highlighting for VXML

use-lexer-xcode ()

Force syntax highlighting for XCode files

use-lexer-xml ()

Force syntax highlighting for XML files

use-lexer-yaml ()

Force syntax highlighting for YAML

zoom-in ()

Zoom in, increasing the text display size temporarily by one font size

zoom-out ()

Zoom out, increasing the text display size temporarily by one font size

Project Manager Commands

These commands act on the project manager or on the current project, regardless of whether the project list has the keyboard focus.

add-current-file-to-project ()

Add the frontmost currently open file to project

add-file-to-project ()

Add an existing file to the project.

add-package-to-project ()

Add a package to the project.

add-tree-to-project ()

Add a whole directory tree to the project.

browse-selected-from-project ()

Browse file currently selected in the project manager

clear-project-main-debug-file ()

Clear main debug file to nothing, so that debugging runs the frontmost window by default

close-project ()

Close currently open project file

compact-project ()

Compact currently open project file by pruning information about non-existent files and non-critical attribs for things like visual state.

debug-selected-from-project ()

Start debugging the file currently selected in the project manager

execute-selected-from-project ()

Execute the file currently selected in the project manager

new-project ()

Create a new project.

open-ext-selected-from-project ()

Open file currently selected in the project manager

open-project ()

Open a project file.

open-selected-from-project ()

Open files currently selected in the project manager

remove-selection-from-project ()

Remove currently selected file or package from the project

save-project ()

Save project file.

save-project-as ()

Save project file under another name.

set-current-as-main-debug-file ()

Set current frontmost file as the main debug file for this project

set-selected-as-main-debug-file ()

Set selected file as the main debug file for this project

show-analysis-stats ()

Show source code analysis statistics

show-project-window ()

Raise the project manager window

use-normal-project ()

Store project in normal format

use-shared-project ()

Store project in sharable format

view-file-properties (loc=None)

View project properties for a particular file (current file is none is given)

view-project-as-flat-tree ()

View project as flattened directory tree from project file

view-project-as-tree ()

View project as directory tree from project file

view-project-by-mime-type ()

View project as tree organized by file mime type

view-project-properties (highlighted_attrib=None)

View or change project-wide properties

Project View Commands

Commands that are available only when the project view has the keyboard focus.

browse-selected-from-project ()

Browse file currently selected in the project manager

debug-selected-from-project ()

Start debugging the file currently selected in the project manager

execute-selected-from-project ()

Execute the file currently selected in the project manager

open-ext-selected-from-project ()

Open file currently selected in the project manager

open-selected-from-project ()

Open files currently selected in the project manager

remove-selection-from-project ()

Remove currently selected file or package from the project

set-selected-as-main-debug-file ()

Set selected file as the main debug file for this project

view-project-as-flat-tree ()

View project as flattened directory tree from project file

view-project-as-tree ()

View project as directory tree from project file

view-project-by-mime-type ()

View project as tree organized by file mime type

Debugger Commands

Commands that control the debugger and current debug process, if any.

break-clear ()

Clear the breakpoint on the current line

break-clear-all ()

Clear all breakpoints

break-clear-clicked ()

Clear the breakpoint at current click location

break-disable ()

Disable the breakpoint on current line

break-disable-all ()

Disable all breakpoints

break-disable-clicked ()

Disable the breakpoint at current click location

break-edit-cond ()

Edit condition for the breakpoint on current line

break-edit-cond-clicked ()

Edit condition for the breakpoint at the current mouse click location

break-enable ()

Enable the breakpoint on the current line

break-enable-all ()

Enable all breakpoints

break-enable-clicked ()

Enable the breakpoint at current click location

break-enable-toggle ()

Toggle whether breakpoint on current line is enabled or disabled

break-ignore ()

Ignore the breakpoint on current line for N iterations

break-ignore-clicked ()

Ignore the breakpoint at the current mouse click location for N iterations

break-set ()

Set a new regular breakpoint on current line

break-set-clicked ()

Set a new regular breakpoint at the current mouse click location

break-set-cond ()

Set a new conditional breakpoint on current line

break-set-cond-clicked ()

Set a new conditionalbreakpoint at the current mouse click location

break-set-temp ()

Set a new temporary breakpoint on current line

break-set-temp-clicked ()

Set a new temporary breakpoint at the current mouse click location

break-toggle ()

Toggle breakpoint at current line (creates new regular bp when one is created)

clear-exception-ignores-list ()

Clear list of exceptions being ignored during debugging

clear-var-errors ()

Clear stored variable errors so they get refetched

collapse-tree-more ()

Collapse whole selected variables display subtree one more level

debug-attach ()

Attach to an already-running debug process

debug-continue ()

Continue (or start) running, to next breakpoint

debug-detach ()

Detach from the debug process and let it run

debug-file ()

Start debugging the current file (rather than the main entry point)

debug-kill ()

Stop debugging

debug-stop ()

Pause free-running execution at current program counter

exception-always-stop ()

Always stop on exceptions, even if they are handled by the code

exception-never-stop ()

Never stop on exceptions, even if they are unhandled in the code

exception-unhandled-stop ()

Stop only on exceptions that are not handled by the code

expand-tree-more ()

Expand whole selected variables display subtree deeper

force-var-reload ()

Force refetch of a value from server

frame-down ()

Move down the current debug stack

frame-up ()

Move up the current debug stack

hide-detail ()

Show the textual value detail area

run-to-cursor ()

Run to current cursor position

show-detail ()

Show the textual value detail area

step-into ()

Step into current execution point, or start debugging at first line

step-out ()

Return from current function

step-over ()

Step over current execution point

watch (style='ref')

Watch selected variable using a direct object reference to track it

watch-expression (expr=None)

Add a new expression to the watch list

watch-module-ref ()

Watch selected value relative to a module looked up by name in sys.modules

watch-parent-ref ()

Watch selected variable using a reference to the value's parent and the key slot for the value

watch-ref ()

Watch selected variable using a direct object reference to track it

watch-symbolic ()

Watch selected value using the symbolic path to it

Debugger Watch Commands

Commands for the debugger's Watch tool (Wing IDE Professional only). These are available only when the watch tool has key board focus.

watch-clear-all ()

Clear all entries from the watch list

watch-clear-selected ()

Clear selected entry from the watch list

Search Manager Commands

Globally available commands defined for the search manager. These commands are available regardless of whether a search manager is visible or has keyboard focus.

batch-replace (look_in=None, use_selection=False)

Display search and replace in files tool.

batch-search (look_in=None, use_selection=True)

Display search in files tool. The look_in argument gets entered in the look in field if not None or ''. The current selection is put into the search field if it doesn't span mulitiple lines and either use_selection is true or there's nothing in the search field.

replace ()

Bring up the search manager in replace mode.

replace-again ()

Replace current selection with the search manager.

replace-and-search ()

Replace current selection and search again.

search ()

Bring up the search manager in search mode.

search-again (search_string='', direction=1)

Search again using the search manager's current settings.

search-backward (search_string=None)

Search again using the search manager's current settings in backward direction

search-forward (search_string='')

Search again using the search manager's current settings in forward direction

search-manager (search_string=None, replace_string=None, action=None, direction=None, auto_search=0, auto_replace=0, auto_replace_all=0, auto_show=0, scope=None, scope_location=None, style=None, match_case=None, whole_words=None, wrap=None, omit_binary=None, flash=False)

Deprecated search command; should not be used in new code.

search-sel ()

Search forward using current selection

search-sel-backward ()

Search backward using current selection

search-sel-forward ()

Search forward using current selection

Search Manager Instance Commands

Commands for a particular search manager instance. These are only available when the search manager has they keyboard focus.

clear ()

Clear selected text

copy ()

Copy selected text

cut ()

Cut selected text

forward-tab ()

Place a forward tab at the current cursor position in search or replace string

paste ()

Paste text from clipboard

License Information

Wing IDE is a commercial product that is based on a number of open source technologies. Although the product source code is available for Wing IDE Professional users (with signed non-disclosure agreement) the product is not itself open source.

The following sections describe the licensing of the product as a whole (the End User License Agreement) and provide required legal statements for the incorporated open source components.

10.1. Wing IDE Software License

This End User License Agreement (EULA) is a CONTRACT between you (either an individual or a single entity) and Wingware, which covers your use of either "Wing IDE Professional" or "Wing IDE Enterprise" and related software components. All such software is referred to herein as the "Software Product." A software license and a license key or serial number ("Software Product License"), issued to a designated user only by Wingware or its authorized agents, is required for each concurrent user of the Software Product. If you do not agree to the terms of this EULA, then do not install or use the Software Product or the Software Product License. By explicitly accepting this EULA you are acknowledging and agreeing to be bound by the following terms:

1. EVALUATION LICENSE WARNING

This Software Product can be used in conjunction with a free evaluation Software Product License. If you are using such an evaluation Software Product License, you may use the Software Product only to evaluate its suitability for purchase. Evaluation Software Product Licenses have an expiration date and most of the features of the software will be disabled after that date. WINGWARE BEARS NO LIABILITY FOR ANY DAMAGES RESULTING FROM USE (OR ATTEMPTED USE AFTER THE EXPIRATION DATE) OF THE SOFTWARE PRODUCT, AND HAS NO DUTY TO PROVIDE ANY SUPPORT BEFORE OR AFTER THE EXPIRATION DATE OF AN EVALUATION LICENSE.

2. GRANT OF NON-EXCLUSIVE LICENSE

Wingware grants the non-exclusive, non-transferable right for a single user to use this Software Product on a single operating system per license purchased. Each additional concurrent user of the Software Product, or each additional operating system where the product is used, requires an additional Software Product License. This includes operating systems on which the Software Product is compiled from source code by the user.

Wingware grants you the right to modify, alter, improve, or enhance the Software Product without limitation, except as described in this EULA.

Although rights to modification of the Software Product are granted by this EULA, you may not tamper with, alter, or use the Software Product in a way that disables, circumvents, or otherwise defeats its built-in licensing verification and enforcement capabilities. The right to modification of the Software Product also does not include the right to remove or alter any trademark, logo, copyright or other proprietary notice, legend, symbol or label in the Software Product.

You may at your discretion distribute patch files containing any modifications or improvements made to the Software Product, other than those that are aimed at disabling or circumventing its built-in license verification capabilities, or that result in the removal or alteration of any trademark, logo, copyright, or other proprietary notice, legend, symbol or label in the Software Product. This right does not include the right to distribute substantial portions of the original source, where distribution rights are limited to contextual information normally existing in software patch files.

You may at your discretion designate license terms, open source or otherwise, for all modifications or improvements made by you. Wingware has no special rights to any such modifications or improvements.

You may make copies of the Software Product as reasonably necessary for its use. Each copy must reproduce all copyright and other proprietary rights notices on or in the Software Product.

You may install each Software Product License on a single computer system. A second installation of the same Software Product License may be made on one other computer system, so long as both copies of the same Software Product License never come into concurrent use. You may also make copies of the Software Product License as necessary for backup and/or archival purposes. Backup and archival copies may not come into active use, together with the Software Product, for any purpose. No other copies may be made. Each copy must reproduce all copyright and other proprietary rights notices on or in the Software Product License. You may not modify or create derivative copies of the Software Product License.

All rights not expressly granted to you are retained by WIngware.

3. INTELLECTUAL PROPERTY RIGHTS RESERVED BY WINGWARE

The Software Product is owned by Wingware and is protected by United States and international copyright laws and treaties, as well as other intellectual property laws and treaties. You must not remove or alter any copyright notices on any copies of the Software Product. This Software Product copy is licensed, not sold. You may not use, copy, or distribute the Software Product, except as granted by this EULA, without written authorization from Wingware or its designated agents. Furthermore, this EULA does not grant you any rights in connection with any trademarks or service marks of Wingware. Wingware reserves all intellectual property rights, including copyrights, and trademark rights.

4. NO RIGHT TO TRANSFER

You may not rent, lease, lend, or in any way distribute or transfer any rights in this EULA or the Software Product to third parties without Wingware's written approval, and subject to written agreement by the recipient of the terms of this EULA.

5. INDEMNIFICATION

You hereby agree to indemnify Wingware against and hold harmless Wingware from any claims, lawsuits or other losses that arise out of your breach of any provision of this EULA.

6. THIRD PARTY RIGHTS

Any software provided along with the Software Product that is associated with a separate license agreement is licensed to you under the terms of that license agreement. This license does not apply to those portions of the Software Product. Copies of these third party licenses are included in all copies of the Software Product.

7. SUPPORT SERVICES

Wingware may provide you with support services related to the Software Product. Use of any such support services is governed by Wingware policies and programs described in online documentation and/or other Wingware-provided materials.

As part of these support services, Wingware may make available bug lists, planned feature lists, and other supplemental informational materials. WINGWARE MAKES NO WARRANTY OF ANY KIND FOR THESE MATERIALS AND ASSUMES NO LIABILITY WHATSOEVER FOR DAMAGES RESULTING FROM ANY USE OF

THESE MATERIALS. FURTHERMORE, YOU MAY NOT USE ANY MATERIALS PROVIDED IN THIS WAY TO SUPPORT ANY CLAIM MADE AGAINST WINGWARE.

Any supplemental software code or related materials that Wingware provides to you as part of the support services, in periodic updates to the Software Product or otherwise, is to be considered part of the Software Product and is subject to the terms and conditions of this EULA.

With respect to any technical information you provide to Wingware as part of the support services, Wingware may use such information for its business purposes without restriction, including for product support and development. Wingware will not use such technical information in a form that personally identifies you without first obtaining your permission.

9. TERMINATION WITHOUT PREJUDICE TO ANY OTHER RIGHTS

Wingware may terminate this EULA if you fail to comply with any term or condition of this EULA. In such event, you must destroy all copies of the Software Product and Software Product Licenses.

10. U.S. GOVERNMENT USE

If the Software Product is licensed under a U.S. Government contract, you acknowledge that the software and related documentation are "commercial items," as defined in 48 C.F.R 2.01, consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1. You also acknowledge that the software is "commercial computer software" as defined in 48 C.F.R. 252.227-7014(a)(1). U.S. Government agencies and entities and others acquiring under a U.S. Government contract shall have only those rights, and shall be subject to all restrictions, set forth in this EULA. Contractor/manufacturer is Wingware, P.O. Box 1937, Brookline MA 02446-0016, USA.

11. EXPORT RESTRICTIONS

You will not download, export, or re-export the Software Product, any part thereof, or any software, tool, process, or service that is the direct product of the Software Product, to any country, person, or entity -- even to foreign units of your own company -- if such a transfer is in violation of U.S. export restrictions.

12. NO WARRANTIES

YOU ACCEPT THE SOFTWARE PRODUCT AND SOFTWARE PRODUCT LICENSE "AS IS," AND WINGWARE AND ITS THIRD PARTY SUPPLIERS AND LICENSORS MAKE NO WARRANTY AS TO ITS USE, PERFORMANCE, OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, WINGWARE AND ITS THIRD PARTY SUPPLIERS AND LICENSORS DISCLAIM ALL OTHER REPRESENTATIONS, WARRANTIES, AND CONDITIONS, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE PRODUCT REMAINS WITH YOU.

13. LIMITATION OF LIABILITY

THIS LIMITATION OF LIABILITY IS TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW. IN NO EVENT SHALL WINGWARE OR ITS THIRD PARTY SUPPLIERS AND LICENSORS BE LIABLE FOR ANY COSTS OF SUBSTITUTE PRODUCTS OR SERVICES, OR FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, OR LOSS OF BUSINESS INFORMATION) ARISING OUT OF THIS EULA OR THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF WINGWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, WINGWARE'S, AND ITS THIRD PARTY SUPPLIERS' AND LICENSORS', ENTIRE LIABILITY ARISING OUT OF THIS EULA SHALL BE LIMITED TO THE LESSER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR THE PRODUCT LIST PRICE; PROVIDED, HOWEVER, THAT IF YOU HAVE ENTERED INTO AN WINGWARE SUPPORT SERVICES AGREEMENT, WINGWARE'S ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT.

14. HIGH RISK ACTIVITIES

The Software Product is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software Product, or any software, tool, process, or service that was developed using the Software Product, could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). Accordingly, Wingware and its suppliers and licensors specifically disclaim any express

or implied warranty of fitness for High Risk Activities. You agree that Wingware and its suppliers and licensors will not be liable for any claims or damages arising from the use of the Software Product, or any software, tool, process, or service that was developed using the Software Product, in such applications.

15. GOVERNING LAW; ENTIRE AGREEMENT ; DISPUTE RESOLUTION

This EULA is governed by the laws of the Commonwealth of Massachusetts, U.S.A., excluding the application of any conflict of law rules. The United Nations Convention on Contracts for the International Sale of Goods shall not apply.

This EULA is the entire agreement between Wingware and you, and supersedes any other communications or advertising with respect to the Software Product; this EULA may be modified only by written agreement signed by authorized representatives of you and Wingware.

Unless otherwise agreed in writing, all disputes relating to this EULA (excepting any dispute relating to intellectual property rights) shall be subject to final and binding arbitration in the State of Massachusetts, in accordance with the Licensing Agreement Arbitration Rules of the American Arbitration Association, with the losing party paying all costs of arbitration. Arbitration must be by a member of the American Arbitration Association. If any dispute arises under this EULA, the prevailing party shall be reimbursed by the other party for any and all legal fees and costs associated therewith.

16. GENERAL

If any provision of this EULA is held invalid, the remainder of this EULA shall continue in full force and effect.

A waiver by either party of any term or condition of this EULA or any breach thereof, in any one instance, shall not waive such term or condition or any subsequent breach thereof.

17. OUTSIDE THE U.S.

If you are located outside the U.S., then the provisions of this Section shall apply. Les parties aux présentes confirment leur volonté que cette convention de même que tous les documents y compris tout avis qui s'y rattache, soient redigés en langue anglaise. (translation: "The parties confirm that this EULA and all related documentation is and will be in the English language.") You are responsible for complying with any local laws in your jurisdiction which might impact your right to import, export or use the Software Product, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

18. TRADEMARKS

The following are trademarks or registered trademarks of Wingware: Wingware, the dancing bird logo, Wing IDE, Wing IDE 101, Wing IDE Personal, Wing IDE Professional, Wing IDE Enterprise, Wing Debugger, and "Take Flight!".

19. CONTACT INFORMATION

If you have any questions about this EULA, or if you want to contact Wingware for any reason, please direct all correspondence to: Wingware, P.O. Box 1937, Brookline, MA 02446-0016, United States of America or send email to info at wingware.com.

10.2. Open Source License Information

Wing IDE incorporates the following open source technologies, most of which are under OSI Certified Open Source licenses except as indicated in the footnotes:

- atk -- GUI accessibility toolkit by Bill.Haneman, Marc.Mulcahy, and Padraig.Obriain -- LGPL [1]
- docutils -- reStructuredText markup processing by David Goodger and contributors-- Public Domain [2]
- expat -- XML parsing library by the Thai Open Source Software Center Ltd, Clark Cooper, and contributors -- MIT License
- fontconfig -- Font configuration detection and support by Keith Packard -- MIT License
- freetype -- High quality text rendering library by Werner Lemberg, David Turner, and contributors -- FreeType License
- glib -- Object development support library by Hans Breuer, Matthias Clasen, Tor Lillqvist, Tim Janik, Havoc Pennington, Ron Steinke, Owen Taylor, Sebastian Wilhelmi, and contributors -- LGPL [1]
- gtk+ -- Cross-platform GUI library by Jonathan Blandford, Hans Breuer, Matthias Clasen, Tim Janik, Tor Lillqvist, Federico Mena Quintero, Kristian Rietveld, Søren Sandmann, Manish Singh, Owen Taylor, and contributors.-- LGPL [1]
- gtk-engines -- GTK theme engines by The Rasterman, Owen Taylor, Randy Gordon -- LGPL [1]

- gtkscintilla2 -- GTK wrapper for Scintilla by Dennis J Houy, Sven Herzberg, and contributors-- LGPL [1]
- GTK Themes -- Aluminum Alloy by Robert Iszaki (roberTO), AluminumAlloy License [4]; Black-Background based on work by Eric R. Reitz, unspecified [5]; Glider by Link Dupont, LGPL [1]; Glossy P by m5brane, unspecified [5]; gnububble by Kyle Davis, unspecified [5]; H2O by Eric R. Reitz, unspecified [5]; High Contrast, Low Contrast, and Large Print themes by Bill Haneman and T. Liebeck, LGPL [1]; Smokey-Blue by Jakub 'jimmac' Steiner and Paul Hendrick, LGPL [1]; Redmond and Redmond95 by Anonymous, unspecified [5]; Smooth2000 by ajgenius, unspecified [5]; SmoothDesert by Ken Joseph, other [6]; SmoothRetro by Ken Joseph, other [6]; SmoothSeaIce by ajgenius, unspecified [5]
- gtk-wimp -- GTK theme with Windows native look by Raymond Penners, Evan Martin, Owen Taylor, Arnaud Charlet, and Dom Lachowicz.-- LGPL [1]
- libiconv -- Unicode conversion library by Bruno Haible -- LGPL [1]
- libpng -- PNG image support library by Glenn Randers-Pehrson, Andreas Eric Dilger, Guy Eric Schalnat, and contributors -- zlib/libpng License
- libXft -- X windows font rendering by Keith Packard and contributors -- MIT License
- libXrender -- X windows rendering extension by Keith Packard and contributors -- MIT License
- pango -- Text layout and rendering library by Owen Taylor and contributors -- LGPL [1]
- parsetools -- Python parse tree conversion tools by John Ehresman -- MIT License
- py2pdf -- Python source to PDF output converter by Dinu Gherman -- MIT License
- pygtk -- Python bindings for GTK by James Henstridge and contributors -- LGPL [1]
- pyscintilla2 -- Python bindings for gtkscintilla2 by Roberto Cavada and contributors -- LGPL [1]
- python -- The Python programming language by Guido van Rossum, PythonLabs, and contributors -- Python 2.3 License [3]
- render -- Header files for X render extension by Keith Packard -- MIT License
- scintilla -- Source code editor component by Neil Hodgson and contributors -- MIT License

- zlib -- Data compression library by Jean-loup Gailly and Mark Adler -- zlib/libpng License

Notes

[1] The LGPL requires us to redistribute the source code for all libraries linked into Wing IDE. All of these modules are readily available on the internet. In some cases we may have modifications that have not yet been incorporated into the official versions; if you wish to obtain a copy of our version of the sources of any of these modules, please email us at info at wingware.com.

[2] Docutils contains a few parts under other licenses (BSD, Python 2.1, Python 2.2, Python 2.3, and GPL). See the COPYING.txt file in the source distribution for details.

[3] The Python 2.3 license is an OSI Approved Open Source license. Each version of Python is under a similar but unique license; Wing includes only Python 2.3.

[4] Not OSI Approved. Wingware has obtained explicit permission from the author to redistribute these themes.

[5] Not OSI Approved. These GTK themes are widely distributed works that are implicitely in the public domain, but without stated license or copyright. They may be removed from Wing IDE without altering the product's base functionality by removing the correspondingly named directories from bin/gtk-bin/share/themes within the Wing IDE installation.

[6] Not OSI Approved. However, license grants permission to modify and use without limitation.

Scintilla Copyright

We are required by the license terms for Scintilla to include the following copyright notice in this documentation:

```
Copyright 1998-2003 by Neil Hodgson <neilh@scintilla.org>

All Rights Reserved

Permission to use, copy, modify, and distribute this soft-
ware and its
documentation for any purpose and with-
out fee is hereby granted,
provided that the above copyright notice ap-
pear in all copies and that
both that copyright notice and this permission notice ap-
pear in
```

```
supporting documentation.

NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
OR PERFORMANCE OF THIS SOFTWARE.
```

Fontconfig Copyright

We are required by the license terms for Fontconfig to include the following copyright notice in this documentation:

```
Copyright © 2001,2003 Keith Packard

Permission to use, copy, modify, dis-
tribute, and sell this software and its
documentation for any purpose is hereby granted with-
out fee, provided that
the above copyright notice ap-
pear in all copies and that both that
copyright notice and this permission notice appear in support-
ing
documentation, and that the name of Keith Packard not be used in
advertising or publicity pertaining to distribu-
tion of the software without
specific, written prior permission.  Keith Packard makes no
representations about the suitability of this soft-
ware for any purpose.  It
is provided "as is" without express or implied warranty.

KEITH PACKARD DISCLAIMS ALL WARRANTIES WITH RE-
GARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT-
NESS, IN NO
EVENT SHALL KEITH PACKARD BE LIABLE FOR ANY SPECIAL, INDI-
RECT OR
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULT-
ING FROM LOSS OF USE,
DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLI-
GENCE OR OTHER
```

```
TORTIOUS ACTION, ARISING OUT OF OR IN CONNEC-
TION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```